THE KIDNAPPED ROBOT PROBLEM

AS A CLASSIFICATION PROBLEM:

USING ARTIFICIAL NEURAL NETWORKS

TO CHARACTERIZE ROBOT LOCALIZATION

Elizabeth Brennan, B.S.

A Thesis Presented to the Graduate Faculty of
Saint Louis University in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

2019

COMMITTEE IN CHARGE OF CANDIDACY:

Associate Professor Kyle Mitchell,
Chairperson and Advisor
Associate Professor William Ebel
Assistant Professor Srikanth Gururajan

TABLE OF CONTENTS

List of Tables

List of Figures

List of Abbreviations

GPS - Global Positioning System
MCL – Monte Carlo Localization
AMCL – Adaptive or Augmented Monte Carlo Localization
EKF SLAM – Extended Kalman Filter Simultaneous Localization And Mapping
COTS – Commerical Off the Shelf Software
SVM – Support Vector Machine
RGB-D – Red Green Blue-Depth
ROS – Robot Operating System

CHAPTER 1: INTRODUCTION

The Problem

Autonomous mobile robots, from home vacuum cleaners [1] to delivery robots used in hospitals[2] and even the Mars Rovers[3], usually require an estimate of their current location to perform their intended function. With today's technology, Global Positioning System (GPS) is a popular position estimation method for human and robot navigation, but there are frequently scenarios in which GPS is not a viable choice for use with autonomous vehicles. For example, GPS signals cannot penetrate the thick concrete walls of a multilevel structure such as a hospital or school, and signal availability in a downtown environment with a high density of tall buildings is limited. Even if GPS signals are accessible for a robotic application, the GPS position estimates may not be as precise or arrive as frequently as a particular scenario demands[4]. Therefore, autonomous robots require a way to estimate the robot's current position without using a GPS.

The solution is provided by the field of robot localization, which develops algorithms that generate real-time estimates of a robot's current position[5] using a map of the robot's environment, data from the robot's interoceptive and exteroceptive sensors, and motion models derived from the robot's kinematics, or mathematical equations based on the robot's structure that describe the way motion forces change the robot's position[6]. Research into robot localization methods was conducted furiously in the 1990s and early 2000s[5], and many different types of methods were proposed with roots in set theory[7], probabilistic techniques[8], and piecewise functions[9]. The principles of the most popular methods are still used today, augmented by advances in the computational resources of the average robot and higher-quality

sensors that deliver faster, more accurate data than was possible when the algorithms were first developed[10].

Definition of Terms

Monte Carlo Localization

One such method, which became popular for its ease-of-implementation[5], is Monte Carlo Localization (MCL). MCL was proposed in 1999 by Fox, Burgard, Dellaert, and Thrun [8]. MCL performs global localization, meaning that given a map of an environment, it requires no knowledge of the robot's initial position to identify the robot's current position. This is achieved as the robot drives around the environment by observing data and probabilistically estimating the robot's current pose[8]. MCL creates a set of sample poses that represent the possible current locations of the robot. Poses describe the robot's pose in (x, y, and Θ) coordinates. Each pose is assigned a numeric confidence indicator or "weight" that indicates its likelihood to be the true current pose. MCL operation consists of two phases: Robot Motion and Sensor Update[8]. During the Robot Motion phase, poses in the sample set are calculated every time the robot moves by applying the robot's motion model to a subset of the previous sample set's poses. At this time, weights are assigned with a uniform distribution to all poses, or particles, in the sample set. The Sensor Update phase occurs when the robot's exteroceptive sensor, such as a laser scanner or camera, receives new data. The weights assigned to each pose in the sample set are increased based on the degree to which the new sensor data corresponds to the sensor data that was perceived at the sample pose when the environment was first mapped. This weighting process is found in MCL's predecessor, Markov Localization, which also uses weighting to identify samples in the set with the highest likelihood weights as the true current robot location[8]. A resampling process upon robot motion generates a new set of possible robot poses

based on the weight distribution of the sample set from the previous update. Therefore, MCL is also known as the particle filter or "Survival of the Fittest" filter[8].

Variations of Monte Carlo Localization

The algorithm just described is considered the "basic" version of MCL[8] and there are two specific areas of enhancement proposed by its creators that are implemented on a case-by-case basis to improve localization performance[8]. One of the adaptations to MCL is a solution to the scenario in which no pose in the particle filter's sample set corresponds to the robot's true current pose. By always adding to the sample set a few samples with randomly-generated, uniformly-distributed poses, the particle set gains pose diversity that could be of use in the case in which the sample set poses converge to a specific map subspace different than the map subspace that holds the robot's true pose[8]. Another modification of MCL is known as Adaptive[18] or Augmented[8] MCL (AMCL). AMCL seeks to optimize the algorithm's computational resources by changing the number of pose samples in the sample set based on the fewest number of samples that are required to provide accurate pose tracking. When the robot's position is unknown, the algorithm increases the number of particles in the filter to allow more poses to be considered as candidates. When the filter is working properly and the current pose is "approximately known," the algorithm decreases the number of particles in the filter to prevent unnecessary computation[8].

The potential of MCL pose estimates converging to the wrong subset of map space illuminates the difficulty in determining when the particle filter has reached the wrong conclusion about the robot's current location. When the MCL algorithm has converged to an incorrect pose estimate, the filter continues to generate new pose estimates in response to the robot's motion commands, and the filter continues to use sensor data to assign confidence to the poses in the sample set. While the filter may appear to be functioning normally, the sample set

of poses does not contain or does not give proper weight to the robot's true current pose. This phenomenon demonstrates the open problem in mobile robot localization known as the Kidnapped Robot Problem.

The Kidnapped Robot Problem

Campbell et al. describe the Kidnapped Robot Problem to be the scenario in which a mobile robot's "internal position estimate changes significantly but [the robot's] actual location does not" [11]. This exact definition is not agreed upon by all literature on the subject. Some regard the Kidnapped Robot Problem as a purely academic problem used to benchmark a localization algorithm's resilience to fault or the unexpected[12], while others say the Kidnapped Robot Problem occurs frequently for mobile robots, citing examples of home robot vacuum cleaners[1], whose locations are regularly rearranged in the home, or the Robocup soccer robots that are relocated by the referee when they drive out-of-bounds on the soccer field [5]. In addition to the terms used in its definition, situations included in the Kidnapped Robot Problem vary, also.

Categories of The Kidnapped Robot Problem

The only Kidnapping event that all Kidnapped Robot Problem publications include can be described as Human Intervention – that is, a person picks up a robot and puts it down somewhere else. Some researchers like Bukhori et al. [12] distinguish between whether the robot was put down in a place included in the map of the localization environment or in an unmapped location. In Tian et al. [10], this is known as the "carry" method of robot relocation. Tian et al. also include the "push" method – in which the robot is pushed away from its current pose. It should be noted that although this category is described as "human intervention" and examples frequently specify or assume a person is acting on the robot, such events could also be caused by other living things, such as chimpanzees or cats.

Zhang et al. [13], Duan et al.[14], and Tian et al. [10] identify a second category of Kidnapping, which Tian et al. call "localization fault". This category includes mechanical failure, sensor fault, and wheel slip. Mechanical failure and wheel slip events cause the robot to complete motion according to a motion model that is different than the motion model used in the algorithm calculating the robot's localization. When the robot operates according to a motion model different than that with which the robot's motion is being measured, it causes the accuracy of the localization algorithm's estimate of the robot's current position to deteriorate. Sensor fault also deteriorates the accuracy of the localization algorithm's estimates because the incorrect data resulting from the sensor fault misrepresents the view from the robot's current pose in the environment. As this data is fed into the localization algorithm, poor results are produced with time.

Wheel slip is not always regarded as a distinct category of Robot Kidnapping in the literature. Campbell et al. dismiss the importance of monitoring what they refer to as "small-scale kidnapping" such as wheel slip as it is possible that common localization algorithms already have pose correction built-in for these faults[11]. Guyonneau references "robot drift" events, and it can be assumed that wheel slip is included in this classification [7]. Although Zhang et al. consider wheel slip to be a Localization Failure [13], Bukhori et al. include wheel slip in the "mechanical faults" category [12]. This demonstrates that there are close ties between Kidnap Detection and Fault Detection. Like Bukhori et al., who consider wheel slip a "fault", Duan et al. consider events such as "sticking of sensors, slippage of wheels, and noisy readings of internal or external sensors" as robot faults [14]. Campbell et al. see value in detecting wheel slip events because they can be informative about the driving conditions, as the slipping could be

the result of the robot "encounter[ing] an obstacle that it cannot detect, such as a rock abutting one of the wheels" [11].

Campbell et al. [11] consider a third Robot Kidnapping category in addition to Human Intervention and Localization Fault when they include what is closely related to a case known elsewhere[5] as the "Wake-Up Robot Problem". In the Wake-Up Robot Problem, the robot begins its navigation of an environment with an incorrect knowledge of its current position. The localization algorithm is provided a particular starting pose, but the robot's actual starting pose differs in either the (x, y) or orientation coordinates. Starting with a poor initial knowledge introduces this error into subsequent localization activities.

As this discussion demonstrates, the Kidnapped Robot Problem may occur under a variety of conditions, but the principle issue remains that the robot operates under an incorrect knowledge of its position in the environment.

Significance of The Kidnapped Robot Problem

The significance of The Kidnapped Robot Problem lies in the negative consequences a robot faces when operating in the time steps following a Kidnap event. When the Kidnap event is undetected, the robot continues its navigation activities based on an incorrect position estimate. Campbell et al. discuss the risks of operating a robot using an incorrect estimate of the robot's current position[11]. Such a scenario exposes the robot to risks such as driving into mapped hazards that under normal navigation would be detected and avoided with path-planning algorithms. Another risk of continuing to operate a mobile robot after a Kidnapping event includes the likelihood that the robot could become immobilized on obstacles such as walls and corners. Such immobilization is inconvenient for moving the robot, but severe immobilization results in the robot continuing to attempt to execute forward motion commands as it rams into

furniture. This can harm the robot's motors, sensors, and other hardware. Such immobilization cases in home robots often lead to further kidnapping, according to Lee et al.:

Robots are often stuck on something in a manner such that a wheel is off the ground during navigation, and are then carried to another location relatively close to the stuck place by a user [1].

Mobile robotics shares the problem of immobilization events with a more mature field, the automotive industry. In the automotive industry, vehicle immobilization events take the form of a vehicle's wheels spinning to no motion effect, such as when a vehicle is slipping on a patch of ice. The automotive industry has detection mechanisms for vehicle immobilization, but these methods are tailored to larger vehicles than most mobile robot systems[4]. Therefore, mobile robots need their own methods for detecting immobilization events.

The negative effects following a Robot Kidnapping event are not limited to the robot's ability to navigate safely and accurately. Some localization algorithms are designed to revise the map of the environment in an online fashion as the robot navigates, and localization failure could edit the map in undesirable ways and cause "severe map deformation" [11]. The accumulative nature of the Extended Kalman Filter Simultaneous Localization and Mapping (EKF-SLAM) algorithm makes the algorithm particularly susceptible to map deformation, as a kidnapping event deforms all information built prior to the point at which the robot was kidnapped [10]. Path planning algorithms used to create navigation routes that take the robot from its current pose to a goal pose are also hampered by the kidnapping event, as the current pose used in the calculations is no longer accurate and subsequent motion commands will be unlikely to move the robot toward the goal as intended. It is clear that Robot Kidnapping events destroy a mobile robot's

ability to perform its normal duties successfully when the Kidnap events are undetected and not corrected.

CHAPTER 2: REVIEW OF THE LITERATURE

Overcoming The Kidnapped Robot Problem

Fortunately, detection methods for identifying instances of Robot Kidnapping can be implemented in mobile robot systems to prevent the hazards associated with Robot Kidnapping and even improve mobile robot functionality. Once a Kidnapping event has been detected, global pose recovery operations that match the robot's current exteroceptive sensor data to likely locations on the map can be executed to correct the localization algorithm's estimate of the robot's current pose. Kidnapping Detection can be used to gain more information about the current environment or the fault that caused the Kidnapping. A robot path planning algorithm could use the detection of wheel slip faults to plan an alternate route, apply traction control[4], or change the driving strategy[11]. Kidnap Detection could be used to signal that the mobile robot's motion kinematics or sensor data models have been changed. If these changes are faults that occur frequently or predictably, specific motion models or sensor models could be developed to better model robot movement under fault conditions. Then, the Kidnapping Detection method could monitor navigation and decide when to employ the fault models[14]. For localization algorithms that build or modify a map of the environment online as the robot moves, Kidnapping Detection could be an indicator that the sensor data taken after a Kidnapping event should be discarded[11]. The Kidnapped Robot Problem threatens robot safety and map quality, but adequate detection and recovery methods can mitigate the danger and deliver robust mobile robot navigation performance.

Detection Methods for The Kidnapped Robot Problem

Kidnap Detection methods can be generalized into two forms: methods that perform analysis on sensor data and those that analyze properties associated with the localization algorithm during robot movement.

Sensor-Based Kidnap Detection Methods

Additional sensors besides the ones necessary for localization or the robot's function can be added specifically for the purpose of detecting robot Kidnap events. Methods that add sensors for Kidnap Detection have been proposed for specific mobile robot applications, such as using an onboard barometer to sense when the robot has been transported by elevator[15]. Similarly, Lee et al. incorporate a drop switch on home cleaning robots to detect instances in which the robot has been lifted off the floor[1]. Such a solution successfully detects the specific type of kidnapping or localization pitfall for which it was designed. The limitation of this approach is that the methods require a dedicated sensor for every kidnapping classification. Localization errors that cannot be detected by sensors cannot be detected by this method. Adding additional sensors may be impractical for low-power or small form-factor mobile robot applications, or robot platforms that are otherwise resource-limited. Due to these limitations, adding sensors specifically for detecting kidnapping is not a widely applicable or an optimal solution.

The most popular category of sensor-based Kidnapping Detection analyzes raw sensor data independently of the localization algorithm[12] and checks for abnormalities. One benefit of this type of detection scheme is that it can be employed independently of the localization method in use. Many algorithms in this category rely on vision-based localization, such as visual odometry. While visual sensors used in localization are common, other methods that do not use vision sensors, such as scan-matching[12] or range finder data[11], are also common. Basing a Kidnap Detection method on a particular sensor type makes the method unavailable for replication on robots that rely on other types of sensor inputs. Another drawback to analyzing sensor data is that it requires individual programmatic access to raw sensor data channels, which may not be conveniently available for a Commercial Off The Shelf (COTS) robot. Detecting Kidnap events by analyzing raw sensor data also means that the robot's onboard computing

power and resources should be considered to determine whether the machine has the resources to run a sophisticated process, potentially involving computer vision activities such as pointcloud calculations, concurrently with the localization service.

Kidnap Detection is possible with sensor-based methods that add sensors specifically for detecting Kidnap events, or analyze raw sensor data for abnormalities. Another category of Kidnap Detection methods detects kidnap events based on knowledge of the localization method employed on the robot.

Localization-Based Prevention of The Kidnapped Robot Problem

Instead of Kidnap Detection, mobile robot algorithms have been developed to prevent Kidnap events in the first place. Zhang et al. propose a preventative solution to the Kidnapped Robot Problem[4] that adjusts the number of candidate pose estimates in the particle filter set based on the quality of the pose estimate[13]. This enables the localization process to use a small particle set when the pose is relatively known, and have the benefit of pose diversity from a large particle set when the robot pose is unknown. The drawback to this method is that large maps could incur particle set sizes that make it "difficult to implement the algorithm in real time" [13]. Kim et al. intercept the data streams prior to the data being sent to the localization algorithm and improve the quality of the data provided to the localization algorithm, which enables the localization algorithm to generate better estimates[3]. The AMCL localization algorithm's practice of injecting random pose particles into the particle filter is designed to increase sample set diversity serves to prevent Robot Kidnapping [5]. Although these methods are effective at facilitating robust robot localization, the consequences and likelihood of robot kidnapping is often too great for some applications to go without explicit kidnap detection.

Localization-based Detection of The Kidnapped Robot Problem

Localization-based Kidnap Detection methods glean information from the localization process's behavior, to determine whether the robot has been kidnapped. Bukhori et al. monitor "the change in distance to the nearest landmark" within the MCL algorithm to make a Kidnap Detection method that is not dependent on data obtained from vision sensors [12]. Guyonneau proposes a method for evaluating whether the localization algorithm's pose estimate is likely to align with the perceived data[7]. Yi et al.[16] use the particle set weights to compare the similarity of observed data and the map. Using the properties and behavior of the localization algorithm creates an almost seamless addition of Kidnapping Detection to a mobile robot. Additionally, it avoids the intense processing load that a visual data-based detection scheme incurs. However, the methods are only possible if the localization algorithm internals can be accessed by other programs, which may not be the case for a COTS robot. The method proposed by Yi et al. uses the localization algorithm's internal data to detect some categories of Kidnap events, but its performance under wheel slip Kidnap conditions are unknown because this was not evaluated in the study. This calls into question whether the localization algorithm's internal data provides enough information to detect all types of kidnapping or faults. Additionally, Yi et al.'s method requires a global map of the environment, which Bukhori et al. point out is not practical or available in some applications [12]. For methods using particle filters in the localization algorithm, Bukhori et al. voiced a general concern about detecting Kidnap events using the information present in the particle filter [12]. Because particle sets may represent erroneous pose estimates, especially when the robot is navigating areas with many similarities to other parts of the map, the dataset is too easily corrupted to use as the sole source of information for a Kidnapping Detection method. A similar view is expressed by Thrun, Burgard, and Fox in Probabilistic Robotics who note that basing Kidnap Detection on the particle filter's

measurement likelihood, or importance weight, from a single time step is not a good solution because “there exist multiple reasons why the measurement probability may be low, besides a localization failure, ” such as sensor noise or global localization activity[5]. Thrun, Burgard, and Fox suggest tracking the short-term and long-term averages of measurement likelihood, so that the short-term average can be compared to the long-term average at each time step to get a measure of the localization performance.

It should be noted that while sensor-based methods like the drop switch or barometer are straightforward to implement, the other Kidnap Detection methods share a drawback in the effort required for implementation. Such methods require additional processing resources and electrical power to function. The methods also rely on thresholds to evaluate metrics and decide whether the robot has been kidnapped. Metrics or constants used in the methods may be unit-specific, and Campbell et al. note that their Kidnapping classifier is sensitive to changes in the sensor equipment or device hardware and such changes may require adjustments to the constants used in the Kidnap Detection metrics[11]. The numeric values of the thresholds vary by application and by robot configuration, and threshold tuning can be a painstaking process. An additional consideration when using thresholds is that the efficacy of thresholds may vary between indoor and outdoor environments[11], and this could pose challenges for a robot that must navigate multiple environments. When evaluating classifier methods such as a Support Vector Machine (SVM), the quantity of available localization data should be considered. When reviewing their own Robot Kidnapping detection scheme based on an SVM, Campbell et al. note that SVMs usually require large training datasets to achieve acceptable performance and such amounts of data may not be readily available in all robot applications[11].

The problem of Kidnap Detection is the subject of much research, and several approaches have been created to mitigate the likelihood and severity of events. Reviewing the best traits of the discussed methods guides the development of new approaches for responding to Robot Kidnapping events.

Developing an Effective Method of Detection for The Kidnapped Robot Problem

The state-of-the-art in Kidnap Detection schemes results in a collection of properties that can be useful when evaluating new Kidnap Detection methods. A Kidnapping Detection method should have the ability to correctly identify kidnapping events of all kinds – carry, push, wheel slip, and other faults, such as flat tires or immobility[11]. Bukhori et al. describe a successful detection scheme as one that achieves a "high probability of success"[12]. They also note that false positives, or time steps in which the robot is not Kidnapped but the Kidnap Detection scheme incorrectly labels the time steps as Kidnap events, are relatively acceptable because the consequence that follows is low-impact[12]. When a Kidnap Detection algorithm incorrectly labels a time step as a Kidnap event, a global localization process is initiated. While global localization takes somewhat more computational resources than a well-localized localization estimate, it is likely that global localization efforts will improve the robot's current pose estimate and potentially prevent Kidnapping events. For best performance, a Kidnap Detection scheme's computational complexity "should not scale linearly with the area of the mapped environment"[12]. Most importantly, a Kidnap Detection method should detect Kidnapping events in an online or real-time manner in order to identify and mitigate the threat of localization failure as soon as possible[12].

The above criteria based on the state-of-the-art in Kidnap Detection can be used to qualitatively measure the proposed solution for the Kidnapped Robot Problem. Kidnap

Detection schemes can be quantitatively measured by collecting localization data during Robot Kidnapping events, and data collection methods are described next.

Data Collection Methods for The Kidnapped Robot Problem

Data Collection from Robot Trials

To study the Kidnapped Robot Problem and develop detection methods, researchers have to create the conditions of robot kidnapping during robot localization. These conditions can be in the physical world with physical robots, or in the virtual world, with simulated robots and environments. For trials involving robots, most kidnap scenarios are achieved by picking up the robot and carrying it to a different location, as described by Kim et al.[17], Lee et al.[1], and Zhang et al.[13]. Zhang et al. are unique in that when they "Kidnapped" the robot during trials, the robot was always put down in the same direction as the robot's pre-Kidnap motion[13]. The reasoning behind this approach is to simplify the Kidnapping study, but the likelihood of such a real-world kidnapping case is limited. Campbell et al.[11] caused wheel slip events by placing a piece of cardboard under the robot wheels as the robot was pushed to a nearby location to prevent the wheels from moving and recording the motion. This achieves the necessary change in robot pose without registering the robot motion with the robot's wheel encoders, but is not a very realistic approximation of Robot Kidnapping.

Robot Simulations

In the virtual world of computer simulations, robot kidnap events can be manufactured with a robot simulator application. The most common method for Kidnapping a robot is by adding sensor noise to one or all of the sensors used in localization[13]. For example, on a robot that uses a laser range finder and wheel encoders to perform localization, sensor noise can be added to the laser range finder while the usual noise affecting the wheel encoder data remains unmodified. A similar approach used by Duan et al. is to prevent exactly one of the sensors used

in localization from registering movement when it occurs[14]. In the example of the robot with a laser range finder and wheel encoders for odometry, this approach allows the laser range finder data to be processed as normal, but inhibits the data stream from the wheel encoders from being sent to the localization algorithm. The laser range finder data indicates the robot's movement, but the wheel encoders report a stationary robot. In their computer simulations of robot localization, Campbell et al. keep the wheel data unmodified, but samples every fifth pointcloud from the vision sensor, which causes jerky and abrupt changes in the visual sensor data similar to those that occur in kidnap events[11].

Whether collecting data from robots or computer simulations, the effects of Kidnapping events on localization algorithm performance should be represented to the greatest degree of accuracy to achieve the best data sets for studying the Kidnapped Robot Problem. Advances in open-source computer simulations provide enhanced levels of simulation accuracy and make it easy to transfer simulation code onto a robot for further study.

Benefits of Studying the Kidnapped Robot Problem with Computer Simulations

Computer simulations of robot applications improve development efficiency, which saves time and resources when testing robotic applications. Computer simulation is especially beneficial for mobile robot localization experiments because the simulator is an environment that facilitates programmatically repeatable trials and flexible environmental arrangements. Writing code to produce simulations enables Robot Kidnapping scenarios to be tested under repeatable conditions. Simulations can be run overnight, and data can be analyzed in bulk with scripts. The simulator's configurable environments let a researcher add or rearrange furniture in a simulated environment within minutes, instead of having to purchase and move real furniture to test the code with a robot in an office or lab environment. A simulator provides the capability to adjust environment properties at the granular level, such as changing the amount of friction between the

robot wheel and the floor to simulate an ice patch or a cobblestone floor. Such flexibility facilitates exploration of a variety of robot scenarios.

Simulations with Gazebo

The Robot Operating System (ROS) robotics environment has strong technological integration with the open-source Gazebo simulator. Common robots that run ROS, such as the Turtlebot 2 by Willow Garage, often have publically-available robot description files and configuration settings for Gazebo that create an accurate physical and operational representation of the physical robot inside the simulation[18]. These files specify properties such as how each piece of the robot responds to collisions, forces such as gravity, the texture of the environment floor, and others. The level of sophistication in Gazebo allows for highly-realistic modeling of robot behavior. ROS robot models can be added to a Gazebo simulation, and the same code running on the simulated robot can be deployed to a robot running ROS[18].

To support dynamic changes to simulation environment conditions, the Gazebo simulator allows the properties of environments models to be modified during simulation through C++ or Python plugins[18]. Another use of plugins is to "create custom robot hardware interfaces between ros_control [the ROS package for robot driving controls] and Gazebo" [18]. With Gazebo plugins, simulation objects can be referenced by name ("Stoplight1") or by class (all entities of type "Stoplight") to modify the properties of one or many entities. For example, in Gazebo tutorials, plugins are used to set a simulated stoplight's Bulb Color property to green or red to simulate a street intersection[19]. The ease of integration with ROS and the opportunity for customized simulations through programmable plugins make Gazebo a strong simulation tool for complex robot experiments such as those used for researching the Kidnapped Robot Problem.

CHAPTER 3: DEVELOPING A METHODOLOGY FOR RESEARCHING "THE KIDNAPPED ROBOT PROBLEM"

Because of the ease of integration and robust robotics functionality, the Robot Operating System (ROS) robotics platform and the Gazebo simulation tool were chosen to research effective detection of The Kidnapped Robot Problem. The ROS library includes an implementation of the Adaptive Monte Carlo Localization (AMCL) algorithm in its amcl package. The Turtlebot 2 robot produced by Willow Garage runs ROS and includes default connectivity to the *amcl* package, so the ROS platform offered the potential to allow *amcl* simulations to be run on a simulation model Turtlebot 2 robot, and apply the same codebase used in the simulation to the Turtlebot 2 robot for laboratory simulations. The Gazebo simulator effectively models noise in the simulated environment, so there was reasonable hope that data collected from the model robot in the Gazebo simulation could be considered relatively representative of actual robot data.

Before developing a solution for detecting the Kidnapped Robot Problem, a data collection structure had to be developed to generate the quantities of data needed for analyzing patterns related to the Kidnapped Robot Problem. To facilitate data collection, the ROS amcl localization package was modified to log relevant internal data for use in the Kidnap Detection method. A custom Gazebo plugin was developed to inject Kidnapping events into the simulation environment at timed intervals to represent a variety of kidnapping conditions so that data on each kidnapping type could be collected. To automate data collection from the simulation and process the data for analysis, a series of Python and Bash processes were created. Before describing the data collection infrastructure, it is useful to explain the fundamentals of the ROS environment.

ROS Fundamentals

All activity in the ROS operating system is based on message-passing. The ROS Master is a service running on a specific computer network adapter port that coordinates message transmissions across the ROS operating system. All other ROS programs register themselves with the ROS Master as "nodes." Nodes can "subscribe" or "publish" data objects or "messages" to communication channels known as "topics" in order to read or write the data required to perform their activities. Code specifying nodes and node behavior is grouped into ROS "packages".

Packages, nodes, topics, and messages allow programs to work together. The ROS amcl localization package creates an amcl node at start-up that subscribes to ROS topics such as /odom[20]. The /odom topic is published by the ROS navigation stack that converts the robot's wheel encoder sensor data into an odometry estimate of the robot's cumulative distance rolled. The amcl node uses /odom to update its particle filter, and publishes its localization estimate to a topic called /amcl_pose. An application for visualizing robot localization called RViz subscribes to the /amcl_pose topic in order to receive /amcl_pose messages as they are published. The RViz tool combines the /amcl_pose messages with messages received from another ROS topic called the /map topic to plot the robot's current location on the map.

Software and Hardware Environment

Most of the research was conducted from a Windows desktop with Oracle VMWare to run a Ubuntu 14.04 LTS virtual machine. The 64-bit virtual machine had 31.4 GB of RAM and a 3.40 GHz Intel Core i7 processor with 8 cores. The ROS environment installed on the virtual machine was the indigo distribution. The amcl package version was 1.2.13 and the turtlebot_navigation ROS package version was 2.3.7. After the research resulted in a viableKidnap Detection system, the system was copied onto the laptop associated with the

Turtlebot robot. The laptop was a Ubuntu 12.04 (precise) 64-bit machine with 3.7 GB of RAM and a 1.90 GHz Intel Core i3 processor with 4 cores.

amcl Infrastructure

The ROS Adaptive Monte Carlo Localization (amcl) package monitors ROS topics related to the robot's motion and performs robot localization by estimating the robot's current pose in a mapped environment[21]. In the field of robot localization, as in other fields, the use of the word "pose" is intended to mean the combination of the robot's (x, y) position and the robot's orientation Θ. The Turtlebot's exteroceptive sensors for perceiving its external environment include wheel encoders for measuring wheel rotation and a Microsoft Kinect RGB-D camera. These sensors publish data to the /laserscan and /odom topics, and the amcl node subscribes to these topics to obtain the information it needs to update its particle filter. ROS driving programs from packages such as turtlebot_navigation publish messages to the /odom topic as the robot moves[22]. As /odom messages arrive, the AMCL algorithm performs the Robot Motion step of the MCL process and uses the incoming /odom information to update the set of potential current robot poses according to the robot's kinematic motion model. The /laserscan topic relays the laser range finder messages converted from raw image data from the Kinect video camera stream. As new /laserscan messages arrive, amcl performs the Sensor Update portion of the MCL algorithm and re-estimates the confidence weights of the robot's current position.

Goal and Method

It was hoped that learning more about the internal activity of the amcl package algorithm during Robot Kidnapping events would identify certain reproducible amcl behaviors that could be used to detect Kidnap events in the future. The rest of the work is organized as follows: To study amcl's internal activity, it was necessary to modify the amcl codebase so that amcl internal data structures could be published to the ROS environment during robot localization for

collection and analysis. To collect data on the localization process, a custom Kidnapping simulation environment was developed using the Gazebo tool. The Gazebo simulation implements the motion characteristics of the primary Kidnapping categories defined in this work to simulate the effects of each Kidnapping type. The data obtained from the Gazebo scenarios was used for analyzing the effects of the Kidnapped Robot Problem on amcl behavior. A neural network was trained on the data in Kidnapped and non-Kidnapped scenarios, and metrics were run on the detection scheme to evaluate its effectiveness at detecting Robot Kidnapping.

amcl Internal Data Structures

The amcl codebase is written in C++ and available for download from the ROS amcl GitHub page[21]. The amcl package is a series of C++ programs and header files built with the catkin build tool that form data structures and object classes to provide functionality like the particle filter and other structures used in the AMCL localization process. The amcl particle filter object is updated by functions corresponding to the Robot Motion and Sensor Update phases of the AMCL process. A C++ numeric array structure is used to store the current particle filter's (x, y, Θ) poses and the confidence weights associated with each particle filter pose. The properties of these objects are not exposed to the ROS environment by default, so ROS Publisher objects were added to the amcl_node.cpp file of the amcl codebase to publish the particle filter data and enablet further study of the effects of Robot Kidnap events on the particle filter.

Initially, the values of internal data structures were written to text files at each time step instead of being published on a ROS topic. It was later determined that once a detectable pattern was found in the data that could be linked to indicate Kidnap events, it would be necessary to publish this data to a ROS topic so that other ROS nodes involved in Kidnap Detection or Kidnap Recovery could subscribe to the topic and receive the amcl data.

Overlaying the amcl Codebase

In order to publish the values of specific internal variables in the amcl package, it was necessary to modify the amcl package's source code. Research into the subject determined that the best way to modify a ROS package and run the modified version is to create an "overlay," or second installation of the ROS package. The source code for the AMCL package was downloaded from the ROS Github code repository[21]. After obtaining the amcl overlay package, the ROS packages that normally initiated the amcl node had to be redirected to use to the file system location of the new amcl overlay[23]. This was accomplished by using the "source" command within the current command line window and providing the file path to the setup.bash file of the new amcl overlay package. This had the effect of telling the command window that it should use the new amcl overlay package location when the environment receives a request involving the amcl package. To ensure that the environment was set up properly before amcl use, a Bash script called start_amcl.sh was developed to set the amcl overlay source directory in the command line console before the amcl launch file, *amcl_demo.launch,* is called (Appendix C). Researching the process for overlaying the ROS amcl package took significant time and effort, but resulted in a reliable and reproducible way to modify a copy of a ROS package and instruct the ROS environment to use the modified version.

Publishing amcl Algorithm Data to Custom Message Types

Once a process for creating an overlay of the default ROS amcl package was developed, ROS message Publisher objects were added to the amcl overlay codebase for the purpose of collecting data from the amcl algorithm's behavior when subjected to the Kidnapped Robot Problem. The codebase for the ROS amcl package uses ROS Publisher objects to write messages to ROS topics so that ROS nodes can subscribe to the topic and consume its messages.

A Publisher object is assigned a topic name and a ROS message datatype that specifies the topic's message format.

Of initial interest were variables such as the current particle filter's poses and their associated weights, the number of pose clusters present in the particle filter, and the current covariance of the particle filter's pose estimates. The C++ data structures used in the amcl package are datatypes such as floating-point arrays, and such datatypes are not represented in the default ROS message types. The ROS environment allows the development of custom ROS message types [24] and ROS Publishers can publish data in the form of these custom message types. Custom messages for conveying numeric arrays were developed in order to publish the variables used in the amcl overlay codebase onto ROS topics. Thirteen custom ROS message datatypes were created to publish the amcl algorithm's variables on unique ROS topics at each time step (Table C1).

Creating the custom ROS messages required defining ROS message structures in text files in the "msgs" folder of the amcl overlay package. To notify the catkin tool to include the new message files in the amcl overlay environment build path, the message file names were added to the "add_messages" dependencies list in the amcl overlay package's CMakeLists.txt catkin build file. Rebuilding the package with the catkin *catkin_make* command compiled the messages and enabled the new ROS message types to be used in the amcl codebase to publish data. Publisher C++ objects for these newly-created message types were added to the amcl package's amcl_node.cpp file, and the desired amcl information was published to ROS topics for the logging application to consume (Table C2).

During data analysis, it was noticed that the amcl particle filter weights published to the /particlefilter topic from the amcl overlay package had an unexpected uniform distribution. The

original intention of publishing the particle filter weight data was to examine the variety of particle weights present in the particle filter after the Sensor Update step. A question describing the problem was posted to the ROS Answers user forum[25]. Within a few hours, the user community reviewed the question and determined that the particle cloud weights were being published from the part of the amcl algorithm following the resampling step that reduced the weights to a uniform distribution. Once the error was identified, a new topic called /particleweightsPreResample was added to the amcl overlay codebase to publish the particle cloud weights from the Sensor Update step.

Starting amcl in the Gazebo Simulator

Although the amcl codebase required modifications to publish the amcl overlay package's internal data structures to the ROS environment, it was determined that the default amcl launch file[26] for initiating the amcl node from the turtlebot_gazebo ROS package[27] could be used without modification to control amcl localization for the simulated Turtlebot within the Gazebo environment. The turtlebot_gazebo package's amcl demo launch file was acceptable to use without modifications because the package is specifically built for running Turtlebot robot simulations in the Gazebo tool, and the amcl_demo.launch file sets the amcl configuration options to values tailored to the Turtlebot's sensors and motion model for optimal amcl performance.

These modifications to the amcl package and the method by which the amcl overlay package was called resulted in the ability to expose amcl algorithm data on ROS topic channels. The published data was made available to other nodes in the ROS environment for logging or monitoring to support the study of the effects of the Kidnapped Robot Problem on the amcl algorithm.

Gazebo Infrastructure

Preparing the Gazebo simulator for collecting amcl data from a Turtlebot model was a multi-phase process that involved verifying that it was possible to write code to simulate Robot Kidnapping events, adapting configuration files to coordinate the Gazebo and amcl environments, and ensuring data quality in Gazebo and amcl.

Simulating a Turtlebot in Gazebo

The turtlebot_gazebo ROS package lets the ROS environment interface with the Gazebo simulation environment and Turtlebot functionality. The turtlebot_gazebo package includes nodes that load a model of the Turtlebot robot into the Gazebo simulation environment and instantiate connections between the Turtlebot model, the ROS operating system, and the Gazebo simulation world. The integration allows the same ROS commands used on the Turtlebot robot to be used on the model Turtlebot in the Gazebo simulation. The Turtlebot's motion in the Gazebo world is controlled by issuing ROS commands with navigation packages such as turtlebot_navigation. The Turtlebot model's sensors perceive the Gazebo simulated environment thanks to set-up in the turtlebot_gazebo package. The Turtlebot model can subscribe to and publish ROS messages to topics within the Gazebo simulation exactly as the Turtlebot robot can in a non-simulated ROS environment. The turtlebot_gazebo package sets up the necessary connections for a seamless integration of the Turtlebot robot into the Gazebo simulation environment.

Extending Gazebo Functionality with Plugins

The Gazebo simulator environment was enhanced with code that created Robot Kidnapping conditions for studying the Kidnapped Robot Problem. Gazebo "plugin" programs can be added to a Gazebo simulation to programmatically change the Gazebo environment models during simulation execution. Plugins are C++ methods that execute as a background

process while a Gazebo simulation runs. Plugins interact with the Gazebo simulation environment by referencing environment variables such as the robot model's properties, locations of robot or furniture models, and the simulated world's physics properties. A Gazebo plugin was developed to change the robot's current pose without sending a driving command. This scenario simulates the fundamental activity underlying the Kidnapped Robot Problem, in which the robot's pose changes in a way that does not correspond to the motion commands issued to the robot.

Modifying a Gazebo Plugin

The Gazebo plugin was written in the call_spawn_delete.cpp C++ program, and stored in a catkin package named make_plugin. After the plugin code was modified, the catkin_make build tool is used to compile the changes to the C++ code and generate an executable used by Gazebo to run the plugin during the simulation. The c*atkin_make* command is run from the root of the make_plugin folder.

Combining Gazebo Plugin Elements

To create Kidnap conditions, the plugin used elements native to the Gazebo and ROS environments, including Gazebo services, Timers, and Callback Functions. Built-in Gazebo services are convenience methods accessible from anywhere in the ROS environment while Gazebo is running. One such service is /gazebo/set_model_state, which resets the pose of objects within the simulated world to a specified (x, y, Θ) pose. Callback functions, or code blocks whose execution can be scheduled by a Timer [28] or triggered by the arrival of topic messages[29], were used to induce Kidnap events at scheduled intervals during the simulation. Callbacks were scheduled to run at set times during the simulation, and when called, the callback function used the /gazebo/set_model_state Gazebo service to change the robot's location in the

environment. This allowed the robot's pose in the simulation environment to be changed without producing driving commands that would update the amcl node's pose estimate.

A crucial component to the success of this method for rendering Robot Kidnapping events in the Gazebo simulator is the way in which the Gazebo simulator handles its clock mechanisms. The Gazebo environment makes a distinction between Wall Time and Simulation Time. Wall Time is the chronological time according to the host machine running the Gazebo simulation. Simulation Time is based on the time at which the simulation instance was initiated. Simulation Time can be paused to temporarily stop the simulation with the this->world->SetPaused(True) method in the roscpp library. Any ROS messages received on ROS topics are queued up while the simulation is paused. Once the Gazebo simulation is "unpaused" and the simulation time clock resumes, the queued-up ROS messages are executed in the order received within the Gazebo simulation [30]. This functionality ensures that the messages such as motion controls sent to the Turtlebot model are not lost when the Gazebo simulation is paused. The pause behavior provides a realistic foundation for creating continuous Kidnap event simulations without losing data. Adjusting the Turtlebot model's location in the Gazebo simulation while the simulation is paused and executing the Turtlebot model's ROS activity in full when the Gazebo simulation is resumed provides a recreation of the mechanics of Robot Kidnapping events.

Replicating Kidnapping Event Fundamentals in the Gazebo Simulator

The properties of the Gazebo environment and the capabilities of Gazebo C++ plugins were combined to simulate the Kidnapped Robot Problem. To simulate Robot Kidnapping in a repeatable fashion, a callback function initiated by a timer was used to cause Kidnap Events at scheduled intervals during the Gazebo simulation (Figure 1).

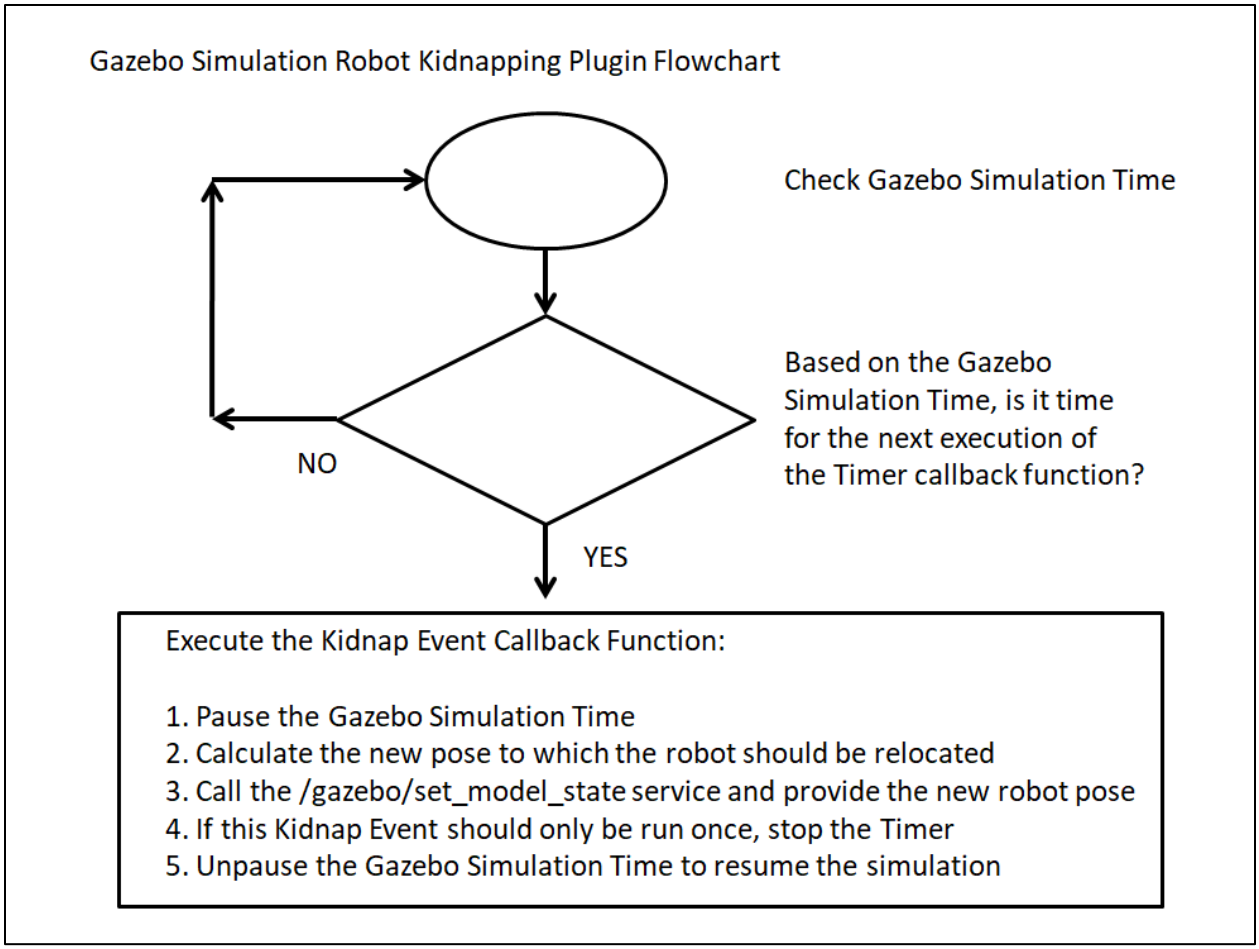


Figure 1: Flowchart for the Gazebo Simulation plugin used to create Kidnap Events

Robot Kidnap events executed by the callback function consisted of relocating the robot model by adjusting the robot model's current pose with the /gazebo/set_model_state Gazebo service, and resuming the simulation. The Timer objects used to schedule the callback functions were executed one or more times, depending on the Kidnap category being simulated. Once initiated, the Timer object executed the callback function at the specified time interval continuously during the simulation until the Timer.Stop() method was called to stop the Timer and prevent the execution of further Kidnap event callback functions. For Kidnap events in which the robot model was to be relocated only once, the Timer was set to execute the Kidnap event callback function after a specified number of seconds following the initiation of the

Gazebo simulation. After the callback function executed, the Timer was stopped with the Timer.Stop() method so that the kidnapping event occurred only once. Prolonged Disturbance events, in which the robot model's motion model during driving fails to match the motion model used in the amcl localization algorithm, were simulated by keeping the Timer object executing repeatedly during the simulation. For each iteration of the Timer object's time duration, the robot model's pose in the Gazebo environment was adjusted according to the callback function's Prolonged Disturbance Kidnapping algorithm.

The work in this phase formalized a method for simulating reproducible instance of the Kidnapped Robot Problem in the Gazebo simulation environment with a simulated Turtlebot robot. To test the accuracy of the Gazebo Robot Kidnapping callback function, 100 samples were collected using the callback function to instantaneously relocate the robot model to the *(x, y, Θ)* pose of (5.5, 5.5, 0.5) as the robot model was driven around the simulated environment. In all 100 samples, the robot model's pose was changed to the exact location of (5.5 meters, 5.5 meters, 0.5 radians) within a maximum of 0.000002 seconds in Wall Time. These results demonstrate the accuracy and reliability of the method used in this research for repositioning the robot model.

Once it was proven that the Gazebo environment could be used to simulate the fundamental behaviors associated with the Kidnapped Robot Problem, the next step was to coordinate the Gazebo simulation environment with the amcl overlay package so that amcl data from Robot Kidnapping events could be collected.

Coordinating amcl and Gazebo

The Gazebo simulation configurations accept a file path to a "world" file that contains all information necessary for creating the Gazebo environment, such as properties specifying the

environment's dimensions, its physics constraints, and the locations and physical representations of object models like furniture or obstacles that are included in the environment. The amcl package configuration options specify a "map" file that documents all sensor data observed while navigating a particular environment, including boundaries of walls and objects encountered in the environment.

Selecting a Simulation Environment World

In order for amcl to accurately estimate the robot's current location in the Gazebo environment, the Gazebo world and the amcl sensor map must be configured with matching environment data. Appendix A details an initial attempt to create a Gazebo world file based on a Saint Louis University classroom and make a sensor map for use with amcl by driving the Turtlebot robot around the classroom. After it was determined that a known amcl bug prevented the custom sensor map from matching the custom Gazebo world file, it was decided that the default Gazebo world and its matching amcl default map would be used for simulating Robot Kidnapping in Gazebo. The *turtlebot_gazebo* package ships with a default amcl sensor map that intentionally matches the Gazebo simulator's default world file. Using the default map for the ROS amcl package and the Gazebo simulator's default world file (Figure 2) provided ease of implementation and ensured agreement between the amcl map and the Gazebo world file. Using the default amcl map and the default Gazebo environment also eliminated the source of error that could have arisen from using a custom amcl map and a custom Gazebo environment developed by first-time ROS users.

The default Gazebo simulated world is a rectangle approximately 54 square meters in size. The default Gazebo simulation world includes several object models that are distributed in a semicircle. Models included in the environment are a jersey barrier, a bookshelf, a cube, a dumpster, and a unit cylinder. The objects are placed about two meters apart around an open

circular area measuring about three meters in diameter. The combination of landmarks and open space provides an ideal setting for a localization test environment because the amcl localization algorithm updates the estimate of the robot model's position in the environment based on which objects are present in the robot model's current field of view. Due to the limited field of view on the Turtlebot model's Kinect sensor, the open area of the Gazebo default world is large enough that the robot model can end up in locations in which the robot model is unable to observe any of the environment objects from its current pose.

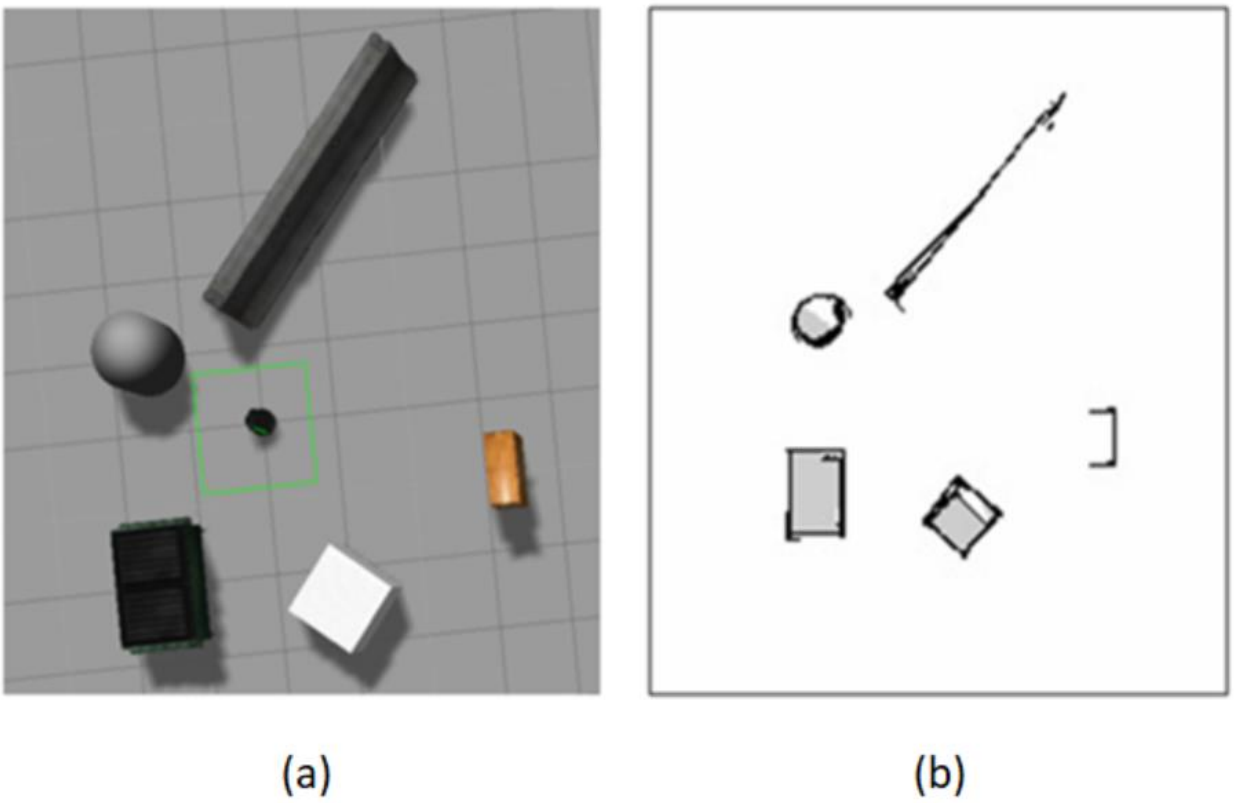


Figure 2: The Gazebo default World (a) has a variety of furniture, such as a cube, a bookshelf, and a dumpster; The gmapping sensor map for the Gazebo default world (b) shows the laserscan outlines of the furniture.

After the amcl map and the Gazebo simulation world were chosen, the next step in coordinating the Gazebo and amcl environments was to ensure that the robot model's initial pose in the amcl algorithm matched the robot model's initial pose in the Gazebo simulation. The

ability of the amcl package to accurately estimate the robot model's position on the map is dependent on the initial robot model pose with which the amcl algorithm is provided. The initial robot model pose estimate can be set in two ways. Commonly, amcl is provided with the robot's current (x, y, Θ) in the Gazebo environment at system start-up. Another way for amcl to obtain its initial robot model pose is by using the amcl package's global_localization service immediately upon amcl startup to distribute the pose particles "randomly through the free space in the map"[21] so that the algorithm can estimate the robot's current pose based on perceived landmarks.

In order to start the simulations of the Kidnapped Robot Problem experiments with minimal error in the initial amcl pose estimate, the Gazebo launch file was edited to specify the initial robot model location to the pose of (0 meters, 0 meters, 0 radians). These values were provided to the amcl configuration file for its *initial_pose_x*, *initial_pose_y*, and *intial_pose_a* arguments. Along the way, it was learned that if it became necessary for the robot model to begin localization from a location other than the Gazebo world origin, the robot model's pose in the Gazebo world could be obtained using the /gazebo/get_model_state service in the ROS environment. The /gazebo/get_model_state service returns the robot model's bearing in quaternion measurements, so before the pose values are provided to the amcl configuration file for the robot model's initial pose, the ROS tf.transformation.euler_from_quaternion service would need to be used to convert the quaternion into radians. Specifying the initial robot model pose in the Gazebo and amcl launch files ensured that Gazebo and amcl were initialized with matching values and created a consistent initialization process used in the Kidnapped Robot

Problem experiments. This helped to ensure the accuracy and quality of the experiment data.

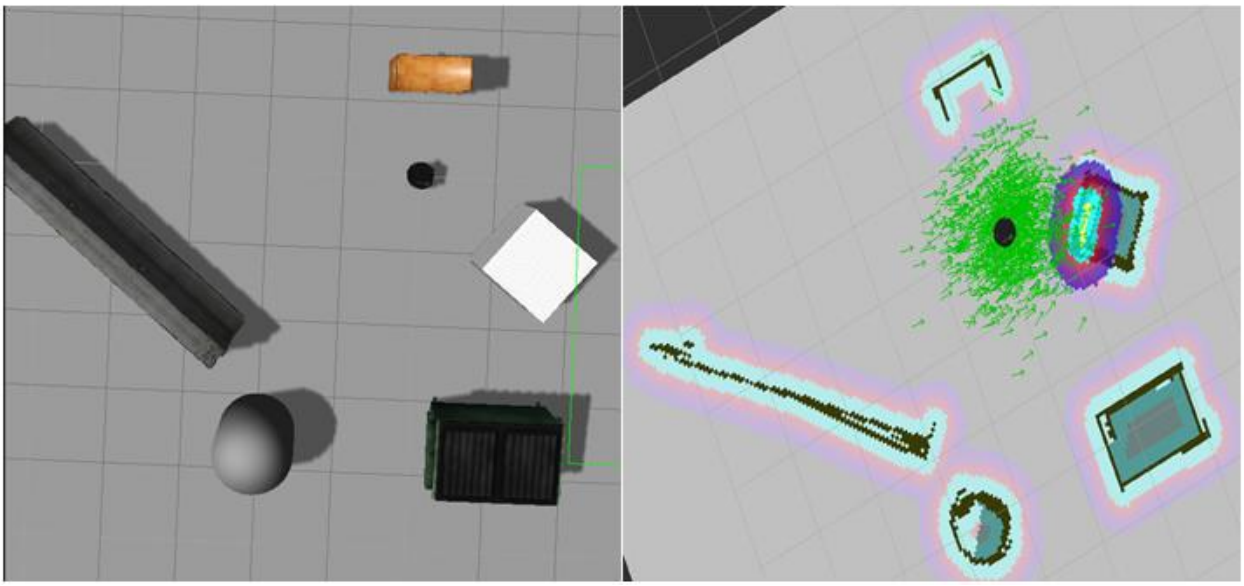


Figure 3: The amcl localization estimate and particle filter on the environment map in RViz compared to the Gazebo simulation

Once the Gazebo and amcl environments were successfully coordinated (Figure 3), methods were reviewed for driving the robot model on a programmable route in the Gazebo simulation environment to enable consistent data collection.

Driving the Robot Model In the Gazebo Simulation

It was deemed important to drive the Turtlebot model on a programmed route in the Gazebo simulations to ensure consistent data collection and facilitate easy comparisons of data from multiple trials. The first approach for programming the robot model's route involved writing a Python script with the rospy ROS client library for the Python programming language to motion controls to the Turtlebot model and drive the robot model around a series of (x, y, Θ) locations in the Gazebo world. When moving the robot model from one pose to the next, the algorithm calculated the necessary angle of approach and drove in a straight line in the approach angle's direction until the destination position was reached. However, lining the robot up to

the proper angle of approach for the next pose often required rotating the robot in place until it reached the specified angle, with some tolerance. This sometimes meant that the robot would pass up the target angle and would spin in circles as it tried again. The motion controls applied to the robot could not be tuned down to small enough increments to prevent this behavior, and increasing the acceptable variance around the angle of approach that the robot was trying to achieve was also unsuccessful. The robot model's spinning-in-place from the driving script also affected the amcl localization accuracy. The amcl algorithm did not track rotating motion very well, and resulted in what was effectively a Robot Kidnapping event of the Wheel Slip variety. Therefore, the Python driving route script did not produce accurate amcl and was not useful for reproducible simulations.

The ROS path-planning algorithms were also considered for programming a driving route for use in the Gazebo simulations. The ROS environment demos use a path-planning framework called MoveIt! to execute motion instructions that move the robot model from its current pose to a goal pose. Implementing the ROS path-planning in the RViz visualization tool for ROS showed that the MoveIt! path-planning results were also subject to frequent rotations, which would again cause undesirable Wheel Slip events.

After the Python driving route script and the ROS MoveIt! path planning algorithm did not produce desired results for driving the robot and conducting localization, the turtlebot_teleop ROS package and the keyboard arrow keys were used to drive the robot around the environment as the rosbag record service recorded the teleop commands. The robot model was first driven around the Gazebo environment with the turtlebot_teleop driving service, which lets the user issue motion commands with the keyboard arrow keys, for 141 seconds, in which the robot travelling 13.8 meters in total. The keyboard-issued driving commands submitted through the

turtlebot_teleop ROS package were recorded as a ROS Bag file. The resulting ROS bag file was used in subsequent trials to play the turtlebot_teleop messages and drive the robot on the recorded route through the Gazebo simulation world (Figure 4). The driving route had to be re-recorded a few times because upon trying to play back the bag file during data collection, it was found that replaying the turtlebot_teleop commands resulted in small execution variations that caused accumulative variance in the robot's route. In some cases, the unexpected routes caused the robot model to become immobilized on environment obstacles. A route was recorded in which the robot model was intentionally steered in a wide path around the environment obstacles to accommodate for the uncertainty in the playback. The robot implements the motion controls with realistic amounts of noise and variation, making the driving route a non-deterministic system.

As a control for the study, ten trials were performed in the Gazebo simulation with no interference with the robot's pose in order to benchmark the robot model's path when driving according to the turtlebot_teleop route recording. Across every time step of the ten trials, the robot's implementation of the Bag recording's turtlebot_teleop driving commands caused the robot's path differing from trial to trial by an average of 1.3 meters. This number was calculated by comparing the Euclidean distance between the robot model's (x, y) poses in Gazebo at each time step for the ten trials.

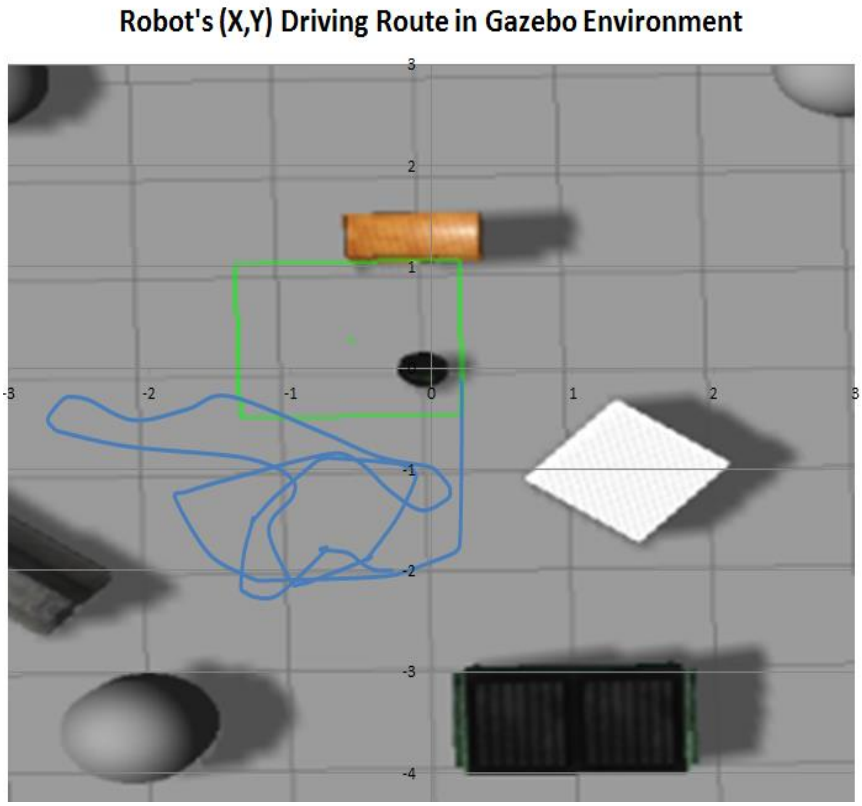


Figure 4: The robot was driven on a programmed route through the simulation environment.

Once it was possible to drive approximately the same route through the Gazebo environment in every simulation, methods were reviewed for recording the data published by the amcl overlay package. A process for collecting data was formalized and implemented.

Data Collection Infrastructure

It was necessary to develop a collection method for saving the data being published by the amcl overlay package. Initially, C++ code was added to the amcl and Gazebo codebases to print log information to text files. As the complexity of the simulation developed, this approach had a limited ability to easily group trial data together or share data with other ROS nodes. The amcl overlay package's codebase and the Gazebo plugin were modified to include ROS Publisher objects for relevant data. To collect the Publishers' data, the ROS environment

includes the rosbag package [31] to record ROS topics and save them into "bag" files with the ".bag" file extension. The rosbag "record" method provided a central way to simultaneously collect ROS topic messages from various sources including the Turtlebot model, Gazebo, and amcl. Topics recorded by the rosbag ROS package during the trials included the custom messages published by the amcl overlay package, messages publishing the robot's current pose in the Gazebo simulation, and other ROS topics such as the /rosout topic, which was used for logging text to the console. See Table B1 for the full list of topics recorded with rosbag record.

In this work, the rosbag process recorded a list of fourteen specific ROS topics and ignored many of the other topics published during the simulation. While many of these topics were extraneous to the focus of the study, later attempts to replay the simulation in its entirety failed because the bag file was missing some of the ROS topics integral to the simulation, such as the /map topic. For future work, all ROS topics available in the system during a simulation should be saved to the bag file to enable to complete replaying of the simulation.

Parsing ROS Bag Files

To perform numeric analysis of message properties from recorded topics, it was deemed beneficial to convert the ROS bag files to Comma Separated Values (CSV) files. Initially, the open-source RosbagPandas Python library[38] was evaluated for performing this task. After attempting to use RosbagPandas on a few trial ROS bag files, it was determined that although the library works for the most common ROS message types, RosbagPandas was not a good tool for the custom datatypes used in the ROS messages recorded from amcl.

To convert the ROS messages in the bag file to a CSV file, a Python program was developed that uses the rospy ROS library to extract ROS message topics from bag files. rospy iteration objects from the ROS Bag class were used to traverse ROS topic message sequences and append the messages to a CSV file. ROS topics were saved to individual CSV files. These

CSV data files included a column for the timestamp of the ROS message's arrival and columns for each of the message object's properties. The CSV files allowed the message data to be easily imported into programs like Microsoft Excel and MATLAB for data analysis. This format was a viable way to study the data because the data structures used by the ROS topic messages were mostly numeric arrays, and as such, the message properties could conveniently be converted to spreadsheet columns. This method could potentially be more difficult to implement for ROS topic messages with complex data structures.

Due to the volume of data generated from the experiments, individual processing of Bag files became unwieldy. A Bash script named parse_bag_files.sh was developed to call the Python parse_bag_file.py method on a series of Bag files for rapid and reliable bulk data processing.

Merging amcl and Gazebo Time Stamps

To match amcl estimates against the Gazebo robot model's pose at the time of the amcl estimate for comparing amcl's localization to the robot model's ground-truth, a program in the R language was developed to join the amcl messages and the Gazebo messages corresponding to the amcl messages' timestamps into a single file. Each trial had on average 129 AMCL messages received from the Publisher objects added to the AMCL codebase. Each amcl message contained an estimate of the robot's pose according to the localization algorithm. The sample rate of AMCL messages during a trial was unpredictable because AMCL pose messages were published as the filter updated the estimates based on motion model updates, caused by the issuance of driving commands, which did not necessarily occur at a standard rate in the driving recording. Messages logging the robot's current pose in Gazebo were published by the base_link topic at a sample rate of about twenty messages per second, causing the average trial to generate 5,687 Gazebo ground-truth pose messages. Gazebo's much higher sample rate meant there was

at least one Gazebo message for every timestamp in the amcl message recordings. Since it was possible to match the amcl time step data to a Gazebo message with a matching timestamp (Figure 5), this method allows the closest possible matching of the Gazebo locations to the amcl pose estimates. amcl timestamps were paired with exactly-matching Gazebo timestamps down to two decimal places, such as 63.12 seconds.

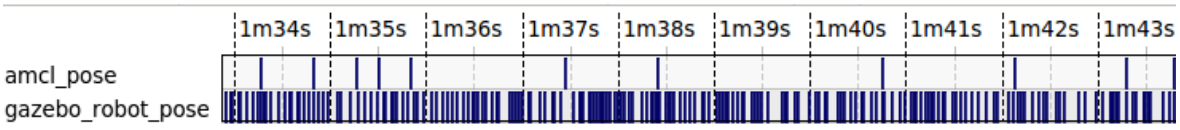


Figure 5: Arrival times of amcl and Gazebo messages relaying the robot's pose estimate and ground-truth pose, respectively

Matching the amcl messages with the Gazebo pose message corresponding to the amcl timestep was an appropriate choice for combining the datasets because although the Gazebo messages arrived at a consistent rate, the amount of motion of the Gazebo robot model between message updates was not consistent. For example, in the 1-second interval between 62 and 63 seconds of Trial 5, the robot's Euclidean distance was adjusted by 0.02 meters. In the next 1-second interval, between 63 and 64 seconds, the robot's pose was adjusted by 0.1 meters. Combining the amcl and Gazebo robot model pose data into a single file allowed for easier comparison of amcl's localization performance with the ground-truth robot model pose.

Running A Simulation

To ensure reproducible experiments, the simulation initialization and data collection processes were standardized with a series of Bash scripts (Appendix C). The scripts contained the ROS commands used to start ROS nodes such as Gazebo and amcl, and included command-line arguments specifying properties such as the Gazebo world file, the amcl map, and the amcl initial location. The rosbag package was invoked using a Bash script to specify the command-line arguments listing the 13 topics that were recorded. Scripting the startup of the ROS nodes

ensured consistency among the data collected by preventing errors such as recording the wrong ROS topic due to a typo or accidentally omitting a command line argument. The scripts were run in individual command prompt windows in the following order:

1. start_gazebo.sh

This process initiates the Gazebo simulation with the specified configuration environment. It should be noted that the Gazebo application does not always launch correctly at this step and sometimes requires manually restarting the Gazebo service to continue. It is a good idea to wait until the Gazebo package prints the "Physics dynamic reconfigure ready." message to the console, as this message indicates that all of the Gazebo start-up processes started successfully. Research into this phenomenon did not turn up any known bug issues or indicate that this is expected behavior. Its root cause is unknown, as the amcl documentation implies that the initialization poses are used exactly as provided. The next script in the sequence, start_amcl.sh, assumes that the Gazebo environment is available, so waiting until the Gazebo process has started before initiating start_amcl.sh ensures that the *amcl* process receives the environmental resources it needs to run smoothly.

2. start_amcl.sh

   This code initiates the AMCL overlay package with the modified code for publishing AMCL internal data. It is a good idea to wait to initiate the next script, start_recording_messages.sh script until the *amcl* package prints the "odom received!" message to the console. This ensures that the AMCL message topics are available when the recording process begins.

3. start_recording_messages.sh

This script creates a Bag file in a location specified by command-line arguments to collect messages from the AMCL and Gazebo processes generated during the simulation.

4. start_driving_robot.sh

   The start_driving_robot.sh code initiates the playing of the Bag file with the recorded Turtlebot driving commands from *turtlebot_teleop*. When the Bag file is finished, start_driving_robot.sh executes a line of code that kills the ROS processes and the Gazebo simulation ("rosnode kill -a" and "killall -9 gzserver"). It should be noted that the Gazebo simulation is not actually ended by this step, and the user should press Control + C to interrupt and end the Gazebo service.

5. *roslaunch turtlebot_rviz_launchers view_navigation.launch*

While not a necessary part of the process, the RViz application is helpful for visualizing the localization efforts of the *amcl* program. RViz plots the particle filter's pose estimates and the motion of the robot model onto the *amcl* sensor map.

Initial Simulation Results

Using the simulation and message recording processes developed in the previous section ("Running a Simulation"), initial robot localization benchmarks were measured to evaluate amcl localization performance in Normal and Kidnapped conditions.

Benchmarking amcl Localization Accuracy in Gazebo Simulation

Benchmark tests were run to evaluate the amcl package's ability to correctly estimate the robot model's pose as it drove around the Gazebo simulation environment, and the results are summarized in Figure 6. The amcl error was calculated by finding the Euclidean distance between the amcl pose estimate and the Gazebo ground-truth of the robot model's pose. The amcl *(x, y)* error averaged 0.37 meters and peaked at 0.5 meters over the 141-second recorded driving route. The amcl pose estimate approximated the robot's Gazebo orientation within an

average of 0.17 radians. This value provides context when evaluating amcl performance during Kidnap events.

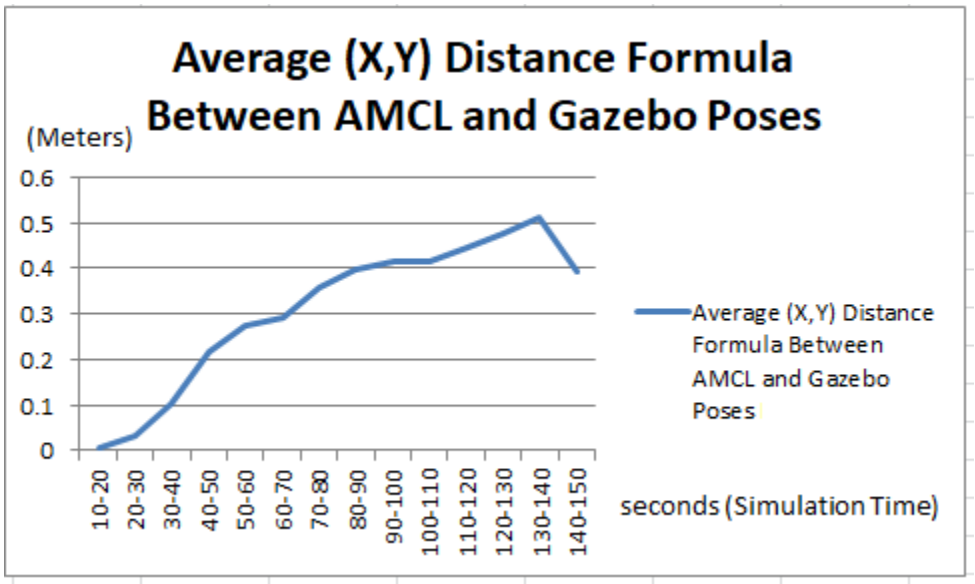


Figure 6: Benchmark Euclidean error results for amcl's estimates of the Gazebo robot model's poses

Figure 7 demonstrates the amcl estimates plotted with the robot model's Gazebo pose.

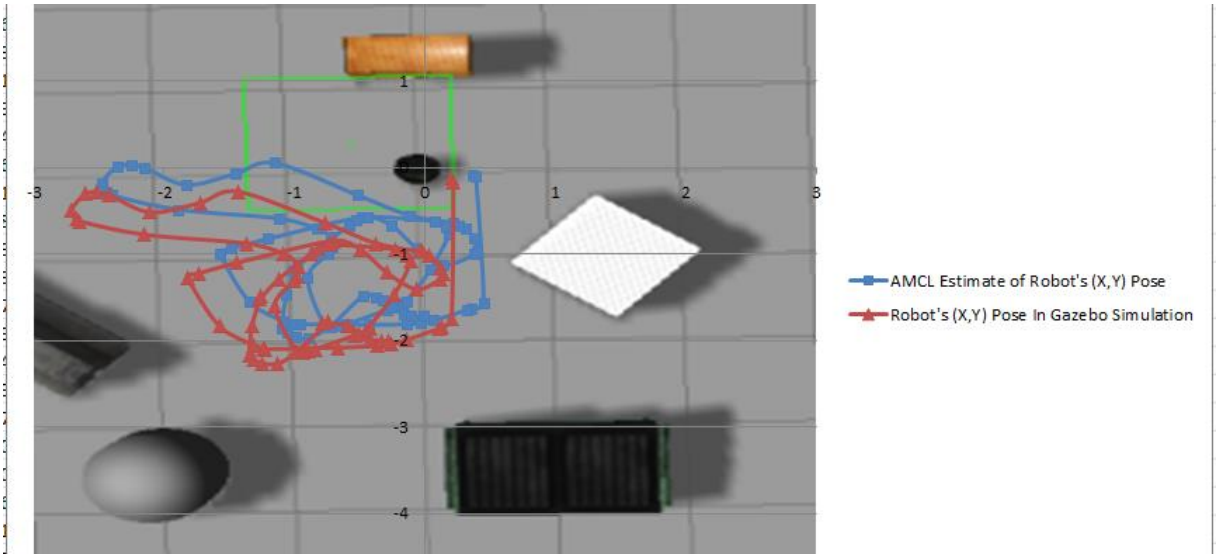


Figure 7: Robot Model's Gazebo Pose compared with the amcl Pose Estimate

Demonstrating Negative Effects on Robot Localization Accuracy Following Robot Kidnapping

The Gazebo plugin was used to inject a Kidnap event at 30 seconds into the Gazebo simulation as the robot model drove around the environment in order to analyze the accumulation of localization error when Robot Kidnapping events go undetected. Over time, the average amcl Euclidean error rose to up to 2 meters (Figure 8).

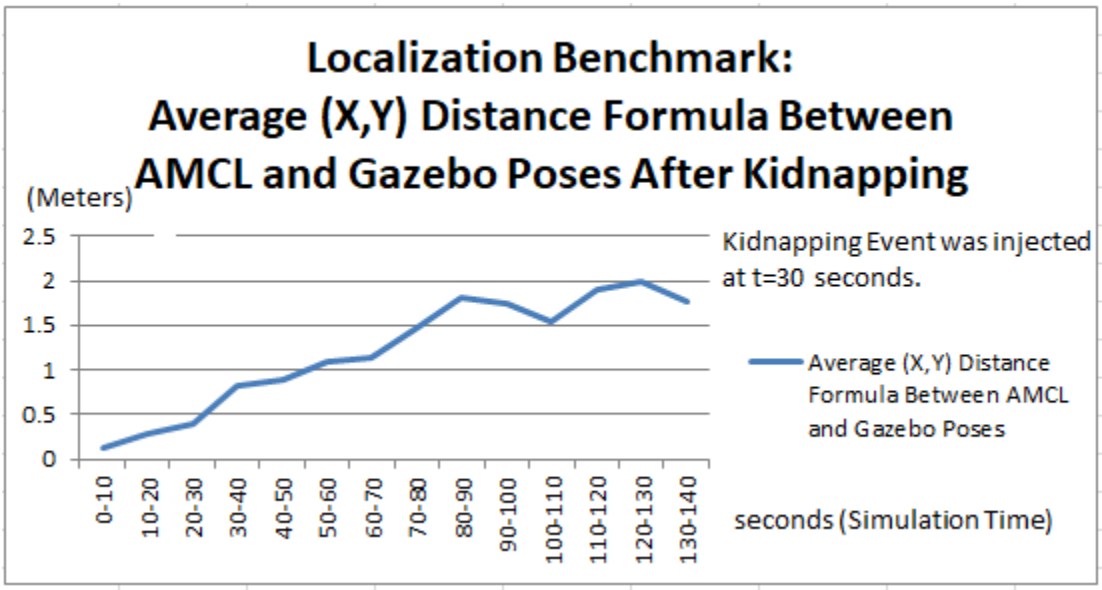


Figure 8: When a Kidnapping Event is undetected, the amcl error rises to up to 2 meters.

Demonstrating Positive Results on Robot Localization Accuracy Following Kidnap Event Detection and Global Localization

To demonstrate that calling the global_localization service to perform global localization following a Kidnapping event to correct the amcl pose estimate, the Gazebo plugin was modified to relocate the robot model to simulate Robot Kidnapping, and then call the global localization service to initiate Kidnap Recovery. The trial data (Figure 9) shows that the amcl pose estimate's Euclidean error for the (x, y) robot model pose in the Gazebo simulation returns to normal levels when global localization follows a Kidnap event.

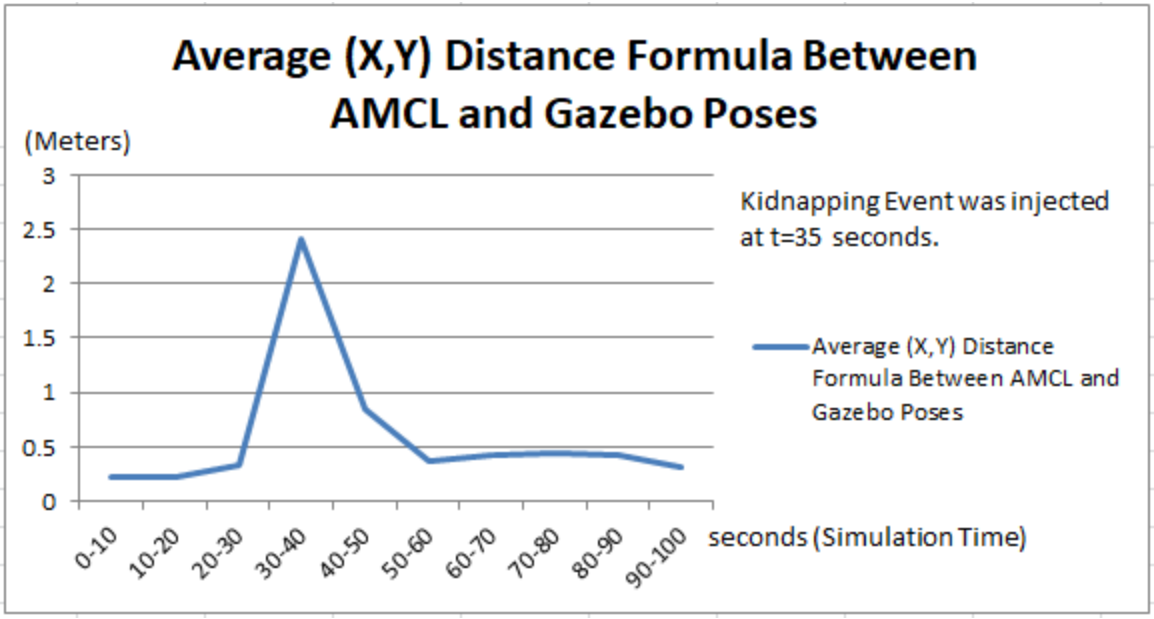


Figure 9: Euclidean error between the amcl pose estimate and the Gazebo robot model pose returns to normal levels following Kidnap Recovery

These benchmarks demonstrated that the amcl algorithm is subject to low levels of error when operating under normal conditions. The Kidnapped Robot Problem's harmful effect on localization accuracy is clear, and global localization is shown to be an effective recovery strategy. The effectiveness of global localization demonstrates the need for Kidnap Detection methods that can initiate global localization upon detection of Robot Kidnapping events.

Modeling Robot Kidnapping Categories

Once the details of the simulation environment and data collection process were finalized, attention turned to expanding the Gazebo plugin to simulate specific Robot Kidnapping conditions. Collecting amcl data following specific categories of Kidnapping events would provide insight into amcl's behavior. In addition to Kidnap Detection, an algorithm that can distinguish between Kidnapping types would be useful for informing driving strategy or path-planning choices.

The Kidnap events most commonly identified in the Kidnapped Robot Problem literature are Carry, Localization Fault, Push, Wheel Slip, Flat Tire, Wake-Up Robot, and Immobility events. Before developing Gazebo plugin functions to simulate each of these categories, the underlying motion models of the categories were reviewed to better understand how to design systems that best approximate the robot motion occurring in the events.

Carry and Wake-Up Robot

The fundamental characteristic of both the Carry and the Wake-Up Robot Kidnapping types is that the robot's pose in the mapped environment at time *t* is different than the expected robot pose. For the Carry event, the robot's pose in the mapped environment at time *t* is different than the robot pose that was expected to have resulted based on the motion controls applied at time (*t*-1).

Wheel Slip, Localization Fault, Push, and Flat Tire

The Wheel Slip, Push, and Flat Tire Kidnap types' fundamental characteristic is that the robot's pose at time *t* is the result of robot motion that is some percentage greater than or less than the robot motion that should have occurred based on the motion controls applied at time (*t*-1). This category includes skid events in which the robot's wheels do not accurately measure the motion that occurred.

If a robot has a flat tire that is losing air as the robot drives, the robot's achieved motion is lessened as the air continuously or sporadically leaves the tire. The air loss causes a progressive decrease in the tire's radius, which results in the robot driving according to a kinematics motion model that differs from the kinematic model used in the localization algorithm to estimate robot motion

In a Wheel Slip Kidnap scenario, the expected friction between the robot wheels and the ground is different than the friction levels taken into consideration by the localization algorithm's kinematic model used for estimating pose updates.

Immobility

The fundamental behavior for Immobility can be likened to an extreme case of Wheel Slip or even a Carry event in the robot is picked up at every timestep and "relocated" to the same pose from which it was picked up.

In the case of Immobility, the robot continues to execute motion controls as assigned, but the robot's wheels cannot achieve the desired motion due to an obstacle or other repelling force and the robot's pose does not change significantly.

Reflection on the Kidnap Types

It should be noted that the distinction between these kidnap types must be decided on a case-by-case basis, as some of the categories overlap or form subsets of one another. If, for example, fault conditions are such that a robot only achieves two percent of its intended motion, there is no set rule that determines whether this scenario is a very dramatic Wheel Slip Kidnapping event or a typical Immobility Kidnapping event.

Underlying all of the Kidnap categories is a singular theme: instead of achieving one hundred percent of the motion specified by the motion controls and kinematics, a robot achieves some fraction of the expected motion, resulting in the effective relocation of the robot. In the Carry Kidnapping event, this relocation could be a one-time event. In the Flat Tire, Immobility, and Wheel Slip types, the kidnap event occurs frequently or continuously during the robot's localization activity.

Generalizing the Kidnapping Types

After evaluating the fundamental characteristics of the Kidnapping events, it was determined that three categories could be used to classify all Kidnapping events based on their fundamental characteristics. Kidnapping events can be categorized as Major Kidnapping events, Minor Kidnapping events, and Prolonged Disturbance Kidnapping events. It is proposed that any Kidnapping scenario in the Kidnapped Robot Problem literature can be represented in one of these basic categories by examining the motion model underlying the Kidnapping scenario (Table 1).

Major Kidnapping events include the Carry and Wake-Up Robot Kidnapping events. Major Kidnapping events are implemented by creating a difference between the robot's current location and its previous location in such a way that the pose difference goes unobserved by the localization algorithm. Minor Kidnapping events include Wheel Slip, Localization Fault, Push, and Flat Tire Kidnapping events. These events are implemented by driving the robot according to a different motion model than the motion model used by the localization algorithm's pose update logic. The Prolonged Disturbance category describes a series of Major Kidnapping or Minor Kidnapping events that occur repeatedly over time, such as the Minor Kidnapping events that occur at every time step as a robot drives a long way on a flat tire or a low battery.

Table 1: Descriptions and Examples of Each Kidnapping Category

Kidnapping Category	**Kidnapping Types Included**	**Description**
Major	Carry Wake-Up Robot Problem, Immobility	Major Kidnapping events are implemented by creating a difference between the robot's current location and its previous location in such a way that the pose difference goes unobserved by the localization algorithm.
Minor	Wheel Slip, Localization Fault, Push, Flat Tire, Immobility	Minor Kidnapping events are implemented by driving the robot according to a different motion model than the motion model used by the localization algorithm's pose update logic.
Prolonged Disturbance	Major Kidnapping events, Minor Kidnapping events, Immobility	Prolonged Disturbance Kidnapping events are a series of Major Kidnapping or Minor Kidnapping events that occur repeatedly over time.

Simulating the Kidnap Types

In the basic Gazebo plugin developed earlier in this work, the Timer callback function simulated Robot Kidnapping by pausing the Gazebo world, using the /gazebo/set_model_state service to change the robot's pose in the simulation environment, and resuming the Gazebo simulation. Once the Kidnapping categories were defined, the Robot Kidnapping method of the Gazebo plugin was enhanced by adding three new functions, one for each Kidnapping category, that calculate the pose to which the robot should be repositioned based on each Kidnapping category's robot motion characteristics. The new Gazebo plugin workflow is: pause the Gazebo world, calculate the pose for the Kidnapping category that is to be simulated with the new functions, set the Gazebo robot model's pose to the new pose obtained from the pose calculation function, and resume the Gazebo simulation. The pose calculation functions used in simulating Major, Minor, and Prolonged Disturbance Kidnapping events are described next.

Function for Major Kidnapping Events

The function calculating the new robot pose following a Major Kidnapping event required the least code changes from the original Gazebo callback function. The Major

Kidnapping function should return a pose that is sufficiently different from the robot's current pose. The robot's pose was set to the arbitrary position in meters of (1, -2) on the map's coordinate plane, with no change to the robot's orientation.

In the Minor Kidnapping and Prolonged Disturbance Kidnapping functions, the Gazebo /get_model_state service is used to obtain the robot model's current pose in the Gazebo environment. The current pose is used to calculate a nearby pose to which the robot model will be relocated. The Minor Kidnapping function takes the current robot model pose and adds some factor in the *x*-, *y*-, or Θ dimension. In one set of Minor Kidnapping simulations, a factor of +0.1 meters was added to the *x* and *y* dimensions of the robot model's current pose. In another set, the Minor Kidnapping pose calculation function added 0.17 radians to the Θ component of the robot model pose. By adding or subtracting a constant from the current robot model pose, the Minor Kidnapping principle of changing the robot's pose to use an unexpected motion model can be replicated in the Gazebo simulation.

The Prolonged Disturbance Kidnapping category's pose calculation function was developed as a geometry-based approach to adjust the robot's pose after a motion update and transfer the robot to a pose that would have resulted if some specified percentage of the intended motion had occurred. The method for determining the robot's pose in the event of a flat tire or slippery floor, in which the robot only achieves some portion of the expected or intended movement, was performed by calculating an artificial pose estimate based on the robot's pose at t, t-1 (Appendix J). See Appendix I for more information about other efforts that were attempted to simulate the effects of Prolonged Disturbance Kidnapping.

In all of the pose calculation functions, the calculated pose is validated to protect against sending null values to the /set_model_state service, and if the calculated pose passes the

validation step, the function returns the calculated pose. For the Major and Minor kidnapping events, the Timer object that triggers the callback function at a specified interval is disabled after the first Kidnapping event so that the Kidnapping activity only occurs once. For the Prolonged Disturbance category, the Timer remains active after the first Kidnap event so that the robot can be repeatedly relocated.

The pose calculations implement the mechanics underlying the Major, Minor, and Prolonged Disturbance Kidnapping categories and make it possible to simulate a variety of robot fault behavior in the Gazebo simulator. The development of this robust Robot Kidnapping simulation environment enabled data trials to be collected on the Kidnapping categories for further analysis.

Results

The initial work resulted in the development of a Gazebo simulation environment that coordinates with a custom ROS amcl overlay package plugin to simulate the common Robot Kidnapping types. The amcl overlay package was modified to facilitate the publishing of internal variables onto ROS message topics, and a process for converting ROS Bag files into CSV files was developed. The Kidnapped Robot Problem scenarios from the literature review were distilled into three general categories, and functions for simulating these Kidnap categories in Gazebo were developed. The most important outcome from this phase of the research was the development of the method for accurately simulating Robot Kidnap events with the Gazebo simulation environment.

Review

A positive aspect of the project is the way in which it makes effective use of custom-built startup scripts to remove the potential of human error in the experiments as much as possible.

The scripts for collecting data and parsing data make it easy to start simulation trials with exactly the same conditions as those used in the previous experiment. The use of ROS Publisher objects to write data to ROS topics from the amcl and Gazebo nodes creates a unified logging environment across ROS processes. In addition to making it easy to record messages with ROS Bag, publishing the data to the ROS operating system makes the data available to other ROS nodes that could be developed to monitor or manage Robot Kidnapping or Fault events.

A weakness of the Robot Kidnapping simulation system is the method of control for the robot's driving route in the Gazebo simulation. A control method is required for repeatable data collection, but the process of driving the robot around and recording its route is not the optimal driving mechanism. It would be better to have a point-to-point control algorithm in place to make the route easier to modify and control. Another weakness of the chosen driving route is that the method required the route to be driven in a wide path around environment obstacles so that when the motion commands were interpreted with execution uncertainty later, the robot would not end up immobilized on an obstacle. The amount of data that could be collected in a trial was negatively impacted by this requirement of keeping a wide path around the environment obstacles when driving the robot. The original goal had been to collect data in the Gazebo simulation for five minutes, but due to the difficulty in getting a driving route in place that allowed robot trials to be conducted with success, the driving sample used was only two minutes and 21 seconds. To accommodate the shorter recording, the Kidnap events were scheduled "early" in the recordings - at 45, 90, and 120 seconds - with the intent to leave enough time to collect sufficient examples of the amcl algorithm's post-kidnap behavior. It was found that programming the plugin to kidnap the robot once it reached a particular pose was not a reproducible method due to the previously-described route variability. Instead, the robot was

kidnapped at specific timestamps that corresponded to the times at which the robot passed through a particular region of the map.

Although a point-to-point control algorithm was developed to be used in this work, testing showed that the rotational commands required to align the robot model's angle with the goal pose created too much noise for amcl to model and resulted in poor amcl localization performance. An enhancement to the simulation method would be to develop a better way of specifying a path through the Gazebo environment that does not interfere with the amcl algorithm's ability to produce accurate pose estimates. This could include enhancing the localization algorithm's ability to track the rotational motion inherent in the path-planning programs. It was found late in this work that the amcl implementation updates the particle filter pose estimates only after a predetermined amount of translational or rotational motion occurs[21]. By default, the filter is updated after $\frac{\pi}{6}$ radians of rotational motion. amcl's poor localization performance under rotational motion could possibly be improved by decreasing the amount of rotational motion required between amcl filter updates. If the rotational localization were improved, it would allow experiments with more precise robot driving methods to be conducted.

Other Uses for the Developed System

The environment configuration processes and data recording methods outlined for simulating Robot Kidnapping demonstrate the method's usefulness as a robot benchmarking framework. The field of Computer Science has a strong history of developing benchmarks for uniform testing of system performance, such as how many prime numbers a processor can generate as a measure of CPU speed. Robotics does not have a long history of developing benchmarks, although the field of industrial manipulators is beginning to develop standard tests for accuracy, reliability, and other metrics. The field of robot localization has no definitive

localization benchmark test, which is partly the source of the variety of research around the Kidnapped Robot Problem and its proposed solutions. This system demonstrates the ability of the Gazebo tool to be used as a robot benchmarking framework, or a robot test framework. Being able to run repeatable tests and record the results can speed up the development cycle and encourage a unified test environment. The Kidnapped Robot Problem is only one example of the problems that could benefit from rigorous simulation and repeatable testing. The Gazebo plugin could be adapted to simulate fault in other robot configurations and operations, such as a manipulator with a faulty arm link or a low battery. Using the Gazebo simulator provides the benefit of a realistic simulation environment and the easy transfer of simulation code onto a robot. The automation in place for data collection in this research provides reliable data quality and improves the quality of the research findings over those performed with robots using data collection techniques subject to human measurement error.

The Robot Kidnapping simulation system developed in this chapter was used to generate datasets for analyzing amcl localization performance under Robot Kidnapping conditions. The behavior of the amcl algorithm variables was profiled for use in training a neural network function to classify Kidnapping events.

CHAPTER 4: METHODOLOGY FOR IMPLEMENTING A CLASSIFICATION SCHEME FOR "THE KIDNAPPED ROBOT PROBLEM"

The Robot Kidnapping simulation system developed in Chapter 3 was used to generate amcl localization performance data from experiments in which the robot model was driven normally and experiments in which the robot model experienced programmed Kidnapping events (Appendix E).

Analyzing the Trial Data for Reproducible Patterns

To develop a detection scheme for the Kidnapped Robot Problem, ROS messages recorded after Robot Kidnapping events were compared to the ROS messages recorded prior to the Kidnapping events to look for changes in the message data values following the Kidnapping event. The CSV files for ROS topic messages recorded during the trials were reviewed with Microsoft Excel, but for some of the larger data elements, such as the particle filter covariance, it was easier to use the ROS rqt application to view the Bag file messages. Most of the amcl data elements recorded during the trials were discarded from further analysis after they demonstrated no response to Kidnapping events. These unused data elements such as the number of samples in the particle filter are included in Table 2.

Table 2: ROS data elements that did not respond to Kidnap Events

ROS Topic	Description
/cluster_count	The number of clusters within the current particle filter.
/delta_pose	The Euclidean distance between the (x,y) position of the current amcl pose estimate and the amcl pose estimate from the previous time step
/filter_covariance	The current covariance of the amcl particle filter
/max_weight_hyp	The array element number of the current particle filter's pose with the greatest confidence weight
/numsamples	The number of samples within the current particle filter

When reviewing the raw data elements did not reveal obvious data value changes in response to Kidnapping events, calculated metrics were developed to test relationships between

data elements. For example, the /amcl_pose ROS topic representing the amcl pose estimate at each time step was compared with the /particlecloud messages representing the particle filter poses from which the amcl_pose had been selected. The Euclidean distance was calculated between the amcl pose and the mean pose in the particle cloud, but this measure did not show a response to Kidnapping events. A similar metric was created to measure the Euclidean distance between the amcl pose and the pose in the particle filter with the second-highest confidence weight. The amcl algorithm chooses the pose associated with the highest confidence weight value in the particle filter to be the best pose estimate, and it was hoped that there might be a revealing ratio between this pose and the pose associated with the second-highest confidence weight in the particle filter. This measure did not produce a different response between normal localization and Kidnapping events.

Metrics comparing data values from sequential time steps were also calculated, such as a metric comparing the Euclidean distance of mean pose represented in the particle filter at time t to the mean pose represented in the particle filter at time (t-1). Another metric reviewed was the arrival times of the amcl messages. None of the developed metrics produced viable responses to Robot Kidnapping events that could be used to differentiate normal localization performance from localization performance following Kidnapping events.

Particle Cloud Visualization in MATLAB

The next approach was to create MATLAB visualizations for plotting the particle cloud (x, y) poses in the hopes that the plots would provide the insight necessary to better understand the datasets. This approach achieved the breakthrough in this research. MATLAB figures were developed for each time step of a localization trial to plot each of the 100 or so (x, y, Θ) location estimates for the robot's current position. The figures showed that the density of the particle

cloud changes from a broad distribution to a tightly-formed cloud as the amcl localization algorithm hones in on a particular area for its pose estimation (Figures 10, 11, and 12).

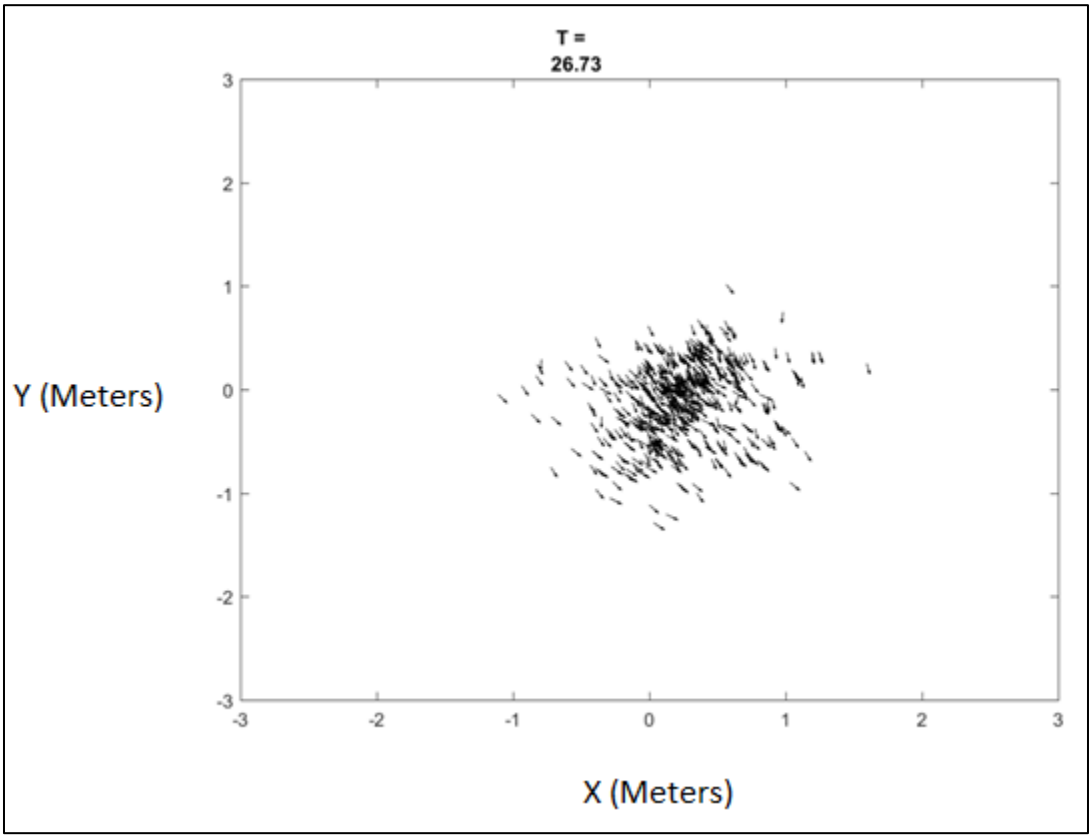


Figure 10: An example of a wide particle cloud

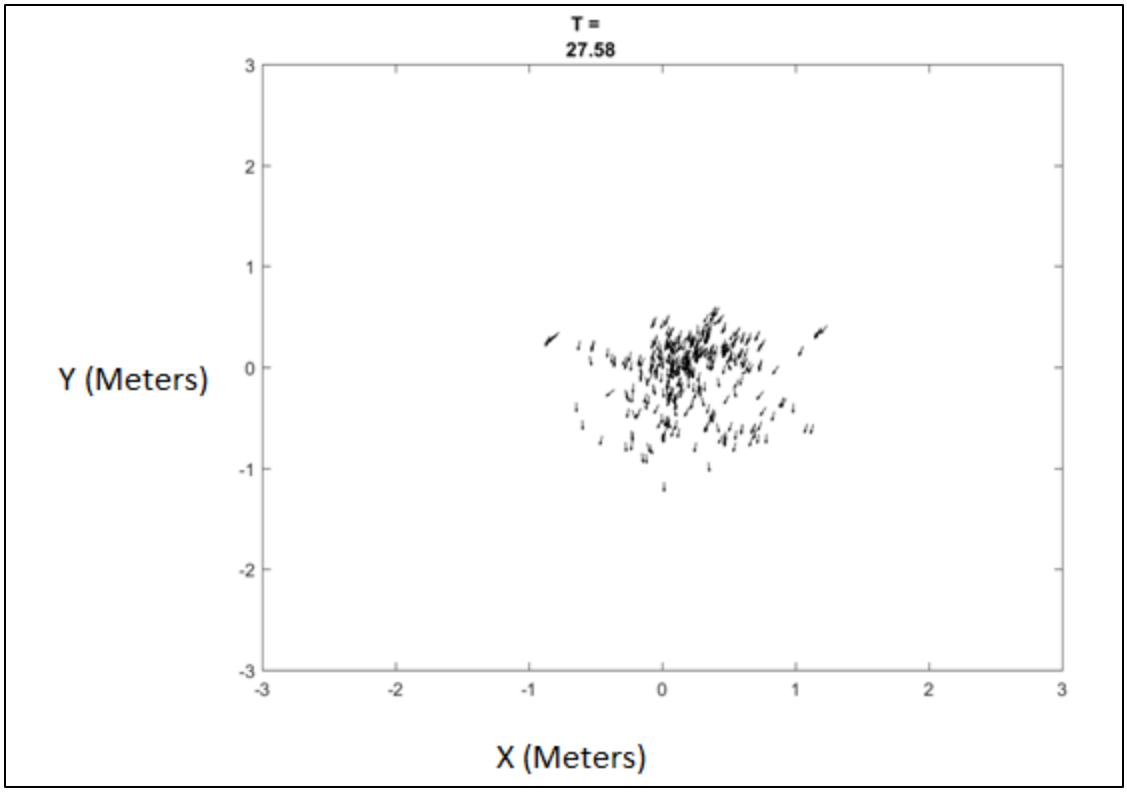


Figure 11: An example of the decreasing breadth of the particle cloud

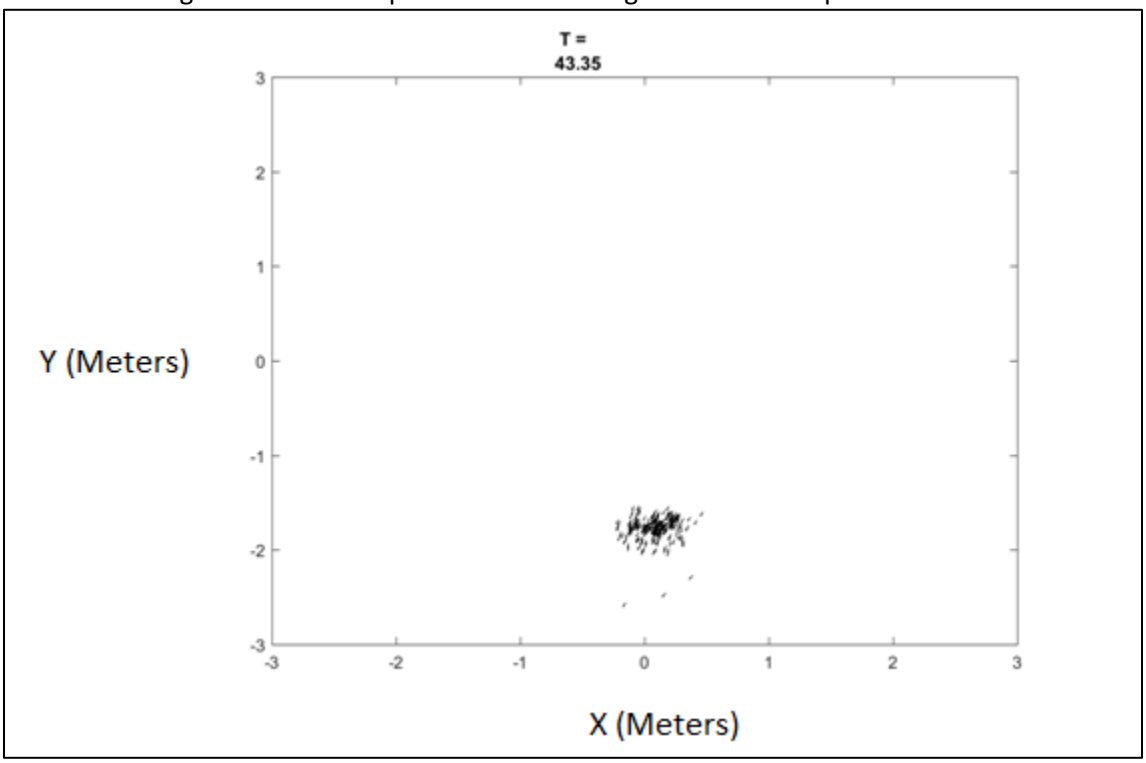


Figure 12: The breadth of the particle cloud collapses as the particle filter's pose estimates become more similar

Metrics for measuring the cloud's diameter was developed by measuring the Euclidean distance between the cloud extremes, but these values did not express a useful response to Kidnapping events.

The MATLAB plots of the particle filter poses were expanded to include the confidence weights associated with each particle filter pose. Color ranges from the default MATLAB colormap known as "parula" were assigned to each(x, y, Θ) vector in the plot based on the value of the confidence weight assigned to the pose estimate vector by the particle filter. To ensure that as many possible particle confidence weights were taken into consideration when the mappings between particle filter confidence weights and MATLAB colors were assigned by the MATLAB default function, the color range calculation was seeded with every time step from every Normal and Kidnapped trial (Figure 13).

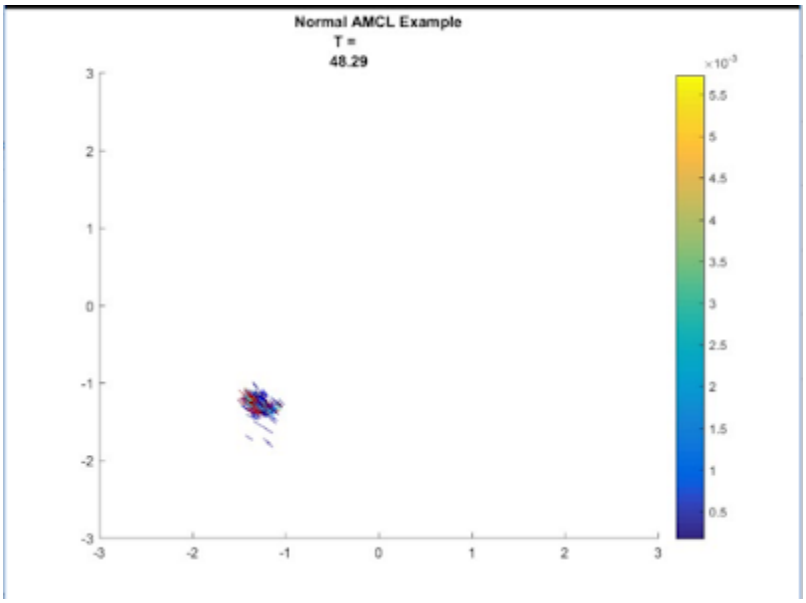


Figure 13: Normal Localization Time Step - Particle Filter Poses are Distributed Across the Color Scheme

When analyzing the chronological progression of the plots of the particle filter poses and confidence weights, it was noticed that the time steps following Kidnapping events demonstrated

a particular color shift in the particle filter vectors (Figure 14). In the time steps following a Kidnapping event, the cloud of pose estimates changes from a multi-colored cloud to a cloud that is entirely bright teal or green. Since the color spectrum assigning color values to the weight values had been created based on a wide variety of localization data sets, it was possible to infer from the plots that the teal or green cloud represented a weight distribution present in the particle filter after Kidnapping events that is not present in the normal localization particle filter.

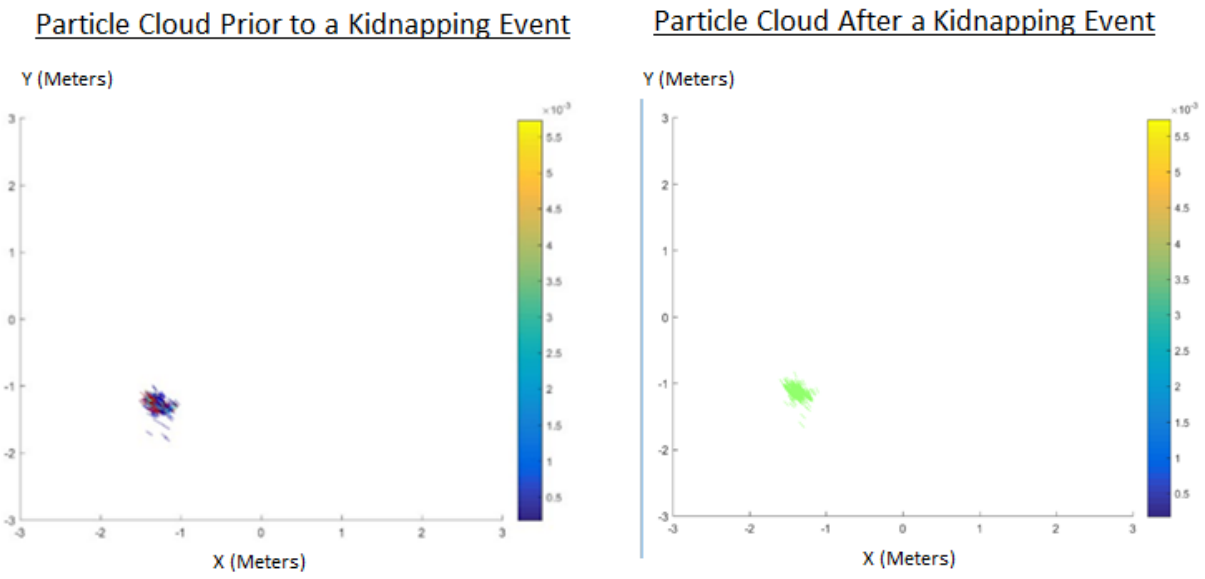


Figure 14: Before a Kidnapping event, the particle filter weights are distributed across the color spectrum (left). After a Kidnapping event, the weights shift to green (right)

Inspired by the ten color ranges into which the particles were represented in the visualizations (Table 3), the particle weights were divided into a ten-bin histogram format in order to use a neural network to identify Kidnapping events. By this point, the particle filter's pose estimates were no longer being considered in the model and the weights assigned to the pose estimates were sufficient for predicting Kidnapping.

Table 3: Color Spectrum for Particle Filter Weights Datasets Used by MATLAB

Range Minimum	Range Maximum	Color Mapping
$0.5x10^{-3}$	$1x10^{-3}$	
$1x10^{-3}$	$1.5\ x10^{-3}$	
$1.5\ x10^{-3}$	$2\ x10^{-3}$	
$2\ x10^{-3}$	$2.5\ x10^{-3}$	
$2.5\ x10^{-3}$	$3\ x10^{-3}$	
$3\ x10^{-3}$	$3.5\ x10^{-3}$	
$3.5\ x10^{-3}$	$4\ x10^{-3}$	
$4x10^{-3}$	$4.5x10^{-3}$	
$4.5x10^{-3}$	$5\ x10^{-3}$	
$5\ x10^{-3}$	$5.5\ x10^{-3}$	

After dividing the weights into histogram bins according to the same range scale included in Table 3, the height of the histogram represents how many particle filter pose estimates were associated with a weight in each bin. Using the ten-bin histograms produced a clear and reproducible pattern in the representation of the particle cloud weight distributions in the aftermath of a Kidnap event. To test the extent of the histogram format, a two-bin histogram scheme was also used to process the data, but this arrangement of the data did not produce a clear pattern that could be used to differentiate the aftermath of a Kidnap event from Normal time steps. It was determined that using ten-bin histograms was the best way to divide the particle weight data so that the difference in the particle filter weight distribution was clearly identified between Normal and Kidnapped datasets.

MATLAB visualizations of the histograms were developed in order to study the data. For visualizing the histograms, the counts for each bin were represented as bin heights. Under regular localization activity, the histogram bin heights show substantial variety in bin heights. This variety of histogram bin heights is not present in the histogram arrangement following a

Kidnapping event, as demonstrated in Figure 11. Under Kidnapping conditions, the bin heights form a spike or skewed-right arrangement that corresponds to the teal or green cloud in the pose estimates plots (Figure 15).

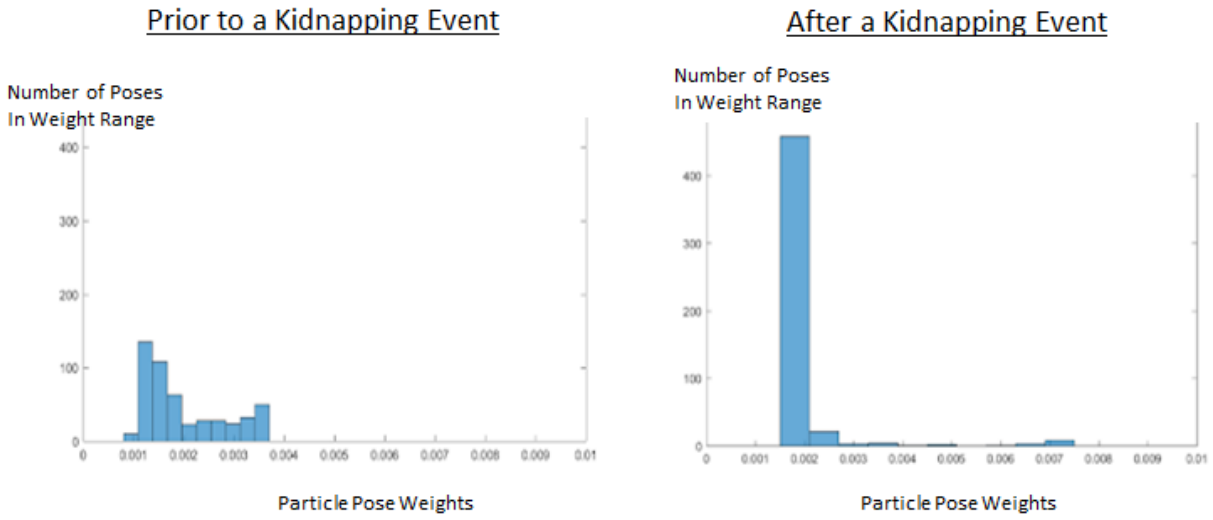


Figure 15: Particle Pose Weight Histogram Examples from a Major Kidnapping Event

Explanation of the AMCL particle cloud weight distributions

The codebase for the ROS amcl algorithm was reviewed to find the root cause of the observed histogram behavior. The amcl code is a direct implementation of the algorithms for probabilistic robot localization derived in Probabilistic Robotics[5], and developer comments included in the amcl code direct the reader to particular pages of the book. The Sensor Update step of ROS amcl localization was examined to determine the causes of the pose importance weight distributions linked to Robot Kidnapping. The enhancement that sets the AMCL algorithm apart from the MCL algorithm is AMCL's use of adding random particles to the particle filter. In amcl, random particles are added to the particle filter "based on some estimate of the localization performance" [5]. The ROS implementation of AMCL in the amcl package generates random particles "in accordance to the measurement distribution" so that the particle poses added to the filter are in locations that correspond to high observation likelihood[5].

To monitor the quality of the algorithm's localization performance, amcl implements the pseudocode of the Augumented_MCL function proposed in Table 8.3 of Probabilistic Robotics [5]. amcl tracks the likelihood that a particular pose in the current particle filter corresponds to the current sensor observations by calculating short-term and long-term averages over each time step, referred to as "fast" and "slow" averages in the ROS amcl codebase. A Kidnap event causes a misalignment between the robot's pose and the sensor observations occurs, causing low likelihoods that the poses in the filter correspond to the sensor observations. As the single misalignment time step becomes several misaligned time steps, the fast average is affected. When the fast average becomes worse than the slow average, the Augmented_MCL algorithm injects random particles into the particle filter to provide pose diversity in the hope that the poses will divert the filter from the area the filter is currently focusing on, which is apparently incorrect. Random samples are added to the particle filter "in proportion to the quotient of the [fast and slow averages]." [5]

The sudden skew to the histograms observed in this research matches the description of AMCL's behavior after "a sudden decay in measurement likelihood," which "induces an increased number of random samples" into the particle filter[5]. From the research in this work, the particle filter's pose importance weights following a Major Kidnapping event form a right-skewed histogram, with a majority of the particle filter weights plotted on the low end of the distribution (Figure 16). The Minor Kidnapping events result in a histogram that is the mirror image of the Major Kidnapping histogram, with a majority of the particle importance weights

plotted on the high end of the distribution.

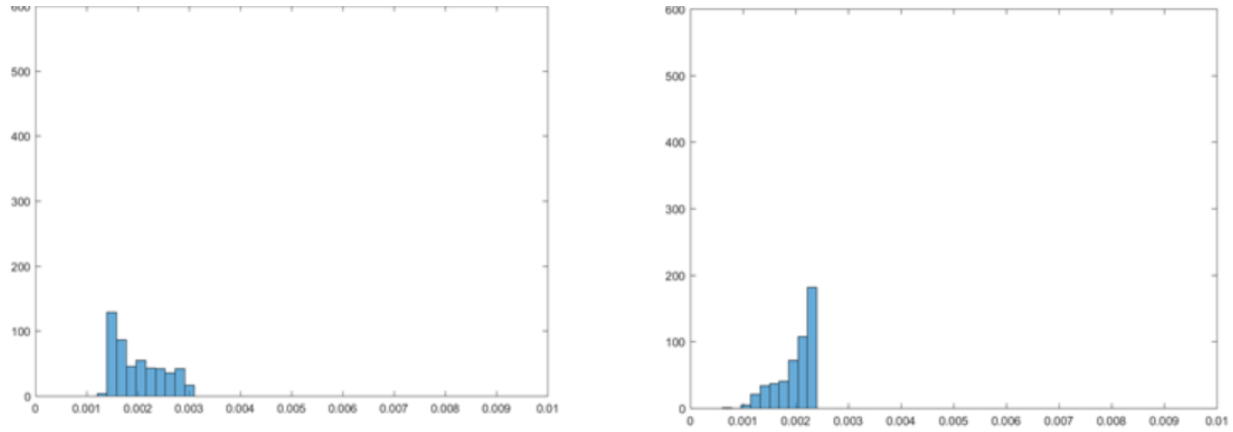


Figure 16: Left: histogram of the particle cloud weights following Major Kidnapping; Right: histogram of the particle cloud weights following Minor Kidnapping

These different patterns to the distributions can be explained by the mechanism used by amcl for setting the importance weights for the random poses added to the particle filter. The code sets the importance weights of the random poses equal to

$$random\_pose\_weight = \frac{weight}{total}$$

where *weight* is 1, as set in line 441 of pf.c, and *total* is the total number of particles in the current particle filter. Since it has been shown that the number of particles in the particle filter is increased by "sudden decay in measurement likelihood" or Robot Kidnapping events, there is a direct association between the importance weights assigned to the random poses and the severity of the Kidnapping event. Major Kidnapping events cause larger disruptions between the fast and slow likelihood averages, and result in greater numbers of random particles added to the particle filter. The increasing number of particles causes the *total* parameter of *random_pose_weight* to grow, and the individual weights of the random particles to shrink. The smaller weights are represented in the histogram plot of the particle cloud weights after a Major Kidnapping event on the lower end of the particle cloud weights. In comparison, Minor Kidnapping events cause less misalignment in the robot's current pose and the current sensor observations, which causes a less

significant difference between the fast and slow averages than the difference of a Major Kidnapping event. The smaller difference in the fast and slow averages causes fewer random particles to be added to the filter. The smaller particle count makes the random_pose_weight calculation assign more weight to the random poses added to the particle filter, resulting in the particles at the high end of the particle cloud histogram.

The connection between the skewed histogram distribution and the fast and slow averages in the AMCL algorithm was discovered late in this research. A better study design would have been to train a neural network on the values of the fast and slow averages at each timestep; as it is, this work's study of the by-product of the fast and slow averages is a sufficient start at classifying Robot Kidnapping based on indicators of pose uncertainty within the AMCL algorithm. It is important to note that the process for adding particles and assigning weight to them based on the number of particles in the filter is not specific to the ROS implementation of the AMCL algorithm in the ROS amcl package codebase. The Augmented_MCL algorithm proposed by Thrun et al. specifies this step, and since the research of Thrun et al. offers the foundation used by most mobile robotics, it is likely - if not mandatory – that other AMCL implementations for other robot frameworks use the same process as this work.

It is worthwhile to discuss the reason that the random particles added to the particle filter in the Adaptive or Augmented Monte Carlo Localization algorithm are unable to correctly estimate the robot's pose after a Kidnapping Event. The step of adding random poses to the particle filter is specifically for making the MCL algorithm robust to Robot Kidnapping[5]. The Augmented MCL proposed in Probabilistic Robotics[5] suggests either choosing random poses in the map or generating the poses by choosing likely map locations based on the sensor observations. The ROS amcl package opts for generating the poses based on the poses included

in the current particle cloud. In the case of Robot Kidnapping events, the robot's post-Kidnap pose is not included in the distribution of poses in the current particle filter. The poses for the random particles added to the particle filter align too closely to the incorrect pose estimate, and the filter never converges to the correct robot pose. The behavior of the ROS amcl package is sufficient for normal robot localization conditions, but an extra system such as the proposed method is necessary for detecting abnormal conditions included in Robot Kidnapping such as wheel slip or Major Kidnapping.

Arranging Trial Datasets to Train the Artificial Neural Network

It was hoped that the histogram bin heights could be used to train an artificial neural network function that could classify a histogram arrangement as either "normal" or "skewed-right." The histogram representations of the particle filter messages were formatted into new CSV files such that a new row in the file was created for each message time step. For each time step row, the height of each histogram bin was recorded in a column, with columns arranged by least-to-greatest particle filter weight from left to right. Each row of the file was also assigned a column for a Kidnapping Flag. The Kidnapping Flag was set to 0 if the localization was running under Normal circumstances, and set to 1 if the time step was associated with or occurred after a Kidnapping event. The first 5 time steps following a Kidnapping event were included in the neural network training datasets, and the time steps from the trial following the 5-time step window were removed as they were deemed corrupted by the Kidnapping event and unreliable.

The neural network was initially trained in MATLAB using the Neural Network Toolbox. Network parameters included a Feed-Forward network with the hidden layer size set to 15 (Figure 17). The default training function of trainlm was used to specify the use of the Leavenberg-Marquardt optimization for the backpropagation function. The learning rate was set

to a constant 0.001, and the number of layers used in the neural network was set to 2. The training data sets were divided into thirds so that the Training, Validation, and Test sets used in the neural network training each received one third of the supplied training data.

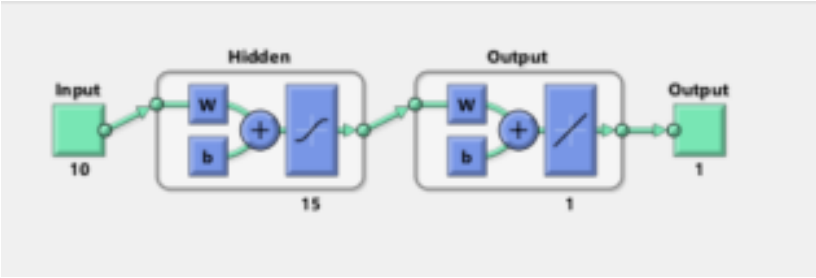


Figure 17: MATLAB Neural Network Toolbox Neural Network Function

Other characteristics of the neural network resulting from training are available in Appendix M.

To test the performance of the function generated by the neural network, the function was run on trial data not included in the test sets and the outputs were compared to the known presence or absence of kidnapping events in the trials' time steps. Appendix E lists the data used to test the neural network classifier.

Analyzing Artificial Neural Network Results

Major Kidnapping

The robot was kidnapped at the 30-, 45-, and 75-second timestamps from its pose to a new pose of the same orientation but with a changed (x, y) position that ranged from 0.3 meters to 2.9 meters in Euclidean distance from the starting robot model pose (Appendix E). It should be noted that the poses to which the robot model was moved were always poses included in the environment maps used by Gazebo and amcl.

The neural network responses for Normal localization trial time steps and time steps following Major Kidnapping events were compared using a Box and Whisker Plot to visualize the statistical distribution of the responses (Figure H1). A Box and Whisker plot displays the

Median, First and Third Quartiles, and Outliers for a given set of data (Appendix G). The neural network function responses following Major Kidnapping events have a mean value that is 1.1 units greater than the mean response of Normal localization trial data (Table K2). This indicates that applying a threshold to the neural network response is likely a useful classifier for deciding whether a Kidnapping event has occurred. To find the optimal threshold value to use to determine whether the robot has been Kidnapped in the current timestamp, the metrics from Table H1 were evaluated. Table K2 shows the performance comparison for each threshold with metrics such as False Positive Rate, False Negative Rate, and Response Time in terms of seconds and amcl update cycles. The threshold with the best False Positive and False Negative rates as well as the fastest Average Time to Detection rate had a 6.55% False Positive rate and a 4.78% False Negative rate. This threshold value is higher than the mean and the 25^{th} percentile, but it is close to the 75^{th} percentile of the Kidnapping response data.

Minor Kidnapping – Orientation:

To test the amcl response to Minor Kidnapping events in which only the robot model's orientation is changed, the robot was Kidnapped at the 30-, 45-, and 75-second timestamps from its pose to a changed pose with the same (x, y) location but with a variation in orientation that ranged from -0.08 to 2.08 radians. Table H3 lists the statistical properties of the neural network output values when presented with Normal and "Minor Kidnapping – Orientation" datasets, and Figure H2 visualizes the results. Table H4 compares the threshold candidates' performance. The best threshold for distinguishing between normal localization data and Kidnapping data is 0.15, representing the minimum response value in the training data of the time steps immediately following the Kidnap event. This threshold achieves a 4.93% False Positive rate and a False Negative Rate of 1.01%. One of the other threshold candidates for this function was the threshold value of 0.2. This threshold had an even lower False Negative rate of 0%, but its False

Positive Rate was 48.46%. When the Median of the neural network responses was evaluated as a threshold, it achieved a lower False Positive rate than the threshold value that was selected, but it also had a higher False Negative rate (5.04%) than the selected threshold. As mentioned earlier, False Positives in Kidnap Detection are of low consequence. It is important to robot localization accuracy that a Kidnap Detection scheme incurs as few False Negatives as possible.

Minor Kidnapping – (X,Y) Pose Adjusted by +0.1 meters

To test the amcl response to Minor Kidnapping events in which only the robot model's (x, y) pose is changed, the robot was kidnapped at the 30-, 45-, and 75-second timestamps from its pose to a changed pose with the same orientation but with a new (x,y) location. For all trials, both the x- and y- coordinate values were increased by 0.1 meters. After calculating threshold candidates (Table H5) and evaluating the thresholds' ability to correctly classify neural network responses (Table H6), it was found that none of the thresholds achieved False Positive Rates lower than 50% accuracy, although the False Negative Rates were no greater than 1% for any of the threshold candidates. The False Positive Rates of the threshold candidates were deemed too high to be implemented. Operating the robot using a Kidnap Detection mechanism with threshold values with False Positive Rates between 50% and 75% would be counterproductive as the global localization process would be called far more often than necessary.

It was proposed that since the simulation involved relocating the robot model by 0.1 meters in the x and y directions, perhaps the Kidnapping event actually at work in this simulation was the Major Kidnapping category. The Major Kidnapping neural network function was applied to the datasets and the thresholds candidates (Table H7) were applied to the neural network response to generate the results in Table K8. The best-performing threshold generated in this method achieved a False Positive Rate of 19.81%, and a 1.89% False Negative Rate.

Combining Models for Classifying Neural Network Output

The best-performing thresholds were assigned for use with the Major and Minor Kidnapping models to classify localization data at each time step as Normal or Kidnapped (Table 4).

Table 4: Threshold Values and Associated False Positive and False Negative Rates By Kidnapping Type

Kidnapping Type	Model	Threshold Value - minimum value of time step following kidnap event	False Negative Rate	False Positive Rate	Average Response Time (Seconds)
Major Kidnapping: On-Map	Major Kidnapping	1.2	4.78%	6.55%	1.12
Minor Kidnapping – XY	Major Kidnapping	0.3992	1.89%	19.81%	2.74
Minor Kidnapping – Orientation	Minor Kidnapping-Orientation	0.15	1.01%	4.93%	1.12

Once the analysis of the model data showed that Kidnap events can be detected with reasonable accuracy, the models were ported into a standalone Kidnap Detection script to be run concurrently with the amcl localization process.

Real-Time Kidnapping Detection with the Turtlebot Robot in Gazebo

In order to perform real-time kidnap detection, it was necessary to create a program that could access live log data from the amcl localization process and analyze the data with the trained neural network function. MATLAB code can be integrated with ROS using several methods. The Robotics Toolbox in MATLAB supports communication with ROS when both the ROS robot and the computer running MATLAB are on the same computer network. The

MATLAB Robotics Toolbox was not easily installed for the MATLAB R2015b version used in this work, so this option was not attempted. Another option is to port the MATLAB code into C++ code and use the ROS C++ library to run the code as a ROS node and communicate with the ROS environment. A complicating factor is that the MATLAB code used in this work involves the MATLAB Neural Network Toolbox dependency. Researching this option revealed that the Neural Network Toolbox dependencies are not easily ported into C++. It was decided that this option was too complex to attempt.

The option that looked the most promising in this analysis was to integrate the neural network functions generated with MATLAB into a Python program so that a neural network could be applied in the same environment in which the rospy ROS library for Python could interact with the ROS robot. The roscpp client library for using ROS in C++ code was another option for integrating a stand-alone neural network function into the ROS environment, but the Python language was chosen because it has a variety of community-developed libraries for machine learning applications. After encountering a variety of challenges with using the Catkin C++ build tool for ROS programming in this research, it was hoped that choosing Python over C++ would make it easier to implement the neural network code. A Python program was developed with a Listener object to receive the cloud_weight message topic logs produced by the amcl ROS process. The cloud_weight localization information received from amcl through the Listener is passed into the MATLAB-trained neural network function, and the neural network function output determines whether the robot is thought to be kidnapped or not. If the robot is thought to be kidnapped, the ROS amcl package's global_localization function can be initiated with rospy to correct and reinitialize the robot's knowledge of its current location.

Reproducing the MATLAB neural network in Python was a challenge. Initially, the MLPClassifier class from the Python SciKitLearn library was trained on the same trial training sets used in the MATLAB environment. The MLPClassifier neural network output did not match that of the MATLAB neural network, so attempts were made to adjust elements of the MLPClassifier to match the MATLAB neural network's configuration. These attempts did not produce the expected neural network outputs and it was found that the MLPClassifier's configuration options did not provide the needed amount of customization to match MATLAB's neural network function specifications. After additional exploration attempts, it was decided that the MLPClassifier was not well-suited for this use-case.

Desired results were finally achieved by programming a neural network in Python "from scratch." The MATLAB genFunction() method was used to produce a report of the MATLAB neural network's layer weights and biases, normalization constants, and the mathematical functions used by the MATLAB neural network. Figure 14 shows the MATLAB neural network process, including:

1. Applying the input layer weights IW1_1 to the input X and adding bias b1,
2. Using the hyperbolic tangent sigmoid transfer function tansig for Layer 1, and
3. Applying the weights LW2_1 and bias b2.

MATLAB Code for The Neural Network Function

```
% Input 1
Xp1 = mapminmax_apply(X{1,ts},x1_step1_gain,x1_step1_xoffset,x1_step1_ymin);

% Layer 1
a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*Xp1);

% Layer 2
a2 = repmat(b2,1,Q) + LW2_1*a1;

% Output 1
Y{1,ts} = mapminmax_reverse(a2,y1_step1_gain,y1_step1_xoffset,y1_step1_ymin);
```

Figure 18: The MATLAB neural network program produced by MATLAB's genFunction () method

The neural network was replicated in Python using the numpy library for arrays and matrix operations support (Figure 18).

```
# Input 1
Xp1 =mapminmax_apply(X,x1_step1_gain, x1_step1_xoffset,x1_step1_ymin)

# Layer 1
a1 = tansig(b1 + np.dot(IW1_1 , Xp1))

# Layer 2
a2 = b2 + np.dot(LW2_1,a1)

# Output 1
return mapminmax_reverse(a2,y1_step1_gain,y1_step1_xoffset,y1_step1_ymin)
```

Figure 19: The Python neural network process that was created to replicate the MATLAB neural network

Python functions were created to match the operations of the MATLAB functions tansig(), mapminmax_apply(), and mapminmax_reverse(), as these methods do not exist in Python or numpy (Figure 20). This step required experimentation to decide which matrix operations, such as the dot product or scalar multiplication, the MATLAB methods were using internally.

```
def tansig(x):
    return 2/(1+np.exp(-2*x)) - 1

def sigmoid(x):
    return 1/(1+np.exp(-x))

def minus(x, settings):
    return x - settings

def times(x, settings):
    return np.multiply(x, settings)

def plus(x, settings):
    return x + settings
def divide(x,settings):
    return x/settings
def divideInt(x,settings):
    return np.divide(x,settings)
def mapminmax_apply(x,settings_gain,settings_xoffset,settings_ymin):
    y = minus(x,settings_xoffset)
    y = times(y,settings_gain)
    y = plus(y,settings_ymin)
    return y
def mapminmax_reverse(x,settings_gain,settings_xoffset,settings_ymin):
    y = minus(x,settings_ymin)
    y =divide(y,settings_gain)
    y = plus(y,settings_xoffset)
    return y

def mapstd_apply(x,input_ps_xmean,input_ps_ystd,input_ps_xstd,input_ps_ymean):
    y = minus(x,input_ps_xmean)
    y2 = divideInt(input_ps_ystd,input_ps_xstd)
    y = times(y,y2)
    y = plus(y,input_ps_ymean)
    return y
```

Figure 20: Python functions developed to recreate MATLAB neural network functionality

The outputs of the custom Python versions of MATLAB functions were compared to the original MATLAB function outputs to ensure that the Python methods were reproducing MATLAB functionality correctly.

The MATLAB neural network constants such as layer weights and biases were specified by the MATLAB genFunction() report, and these values were converted to Python constants. This process involved reformatting the values using the Notepad++ text editor to replace the semicolons used in MATLAB to identify matrix rows with the brackets and commas accepted in Python and numpy. A similar process was used to convert the space-delimited MATLAB arrays

into comma-delimited lists for Python notation. Each kidnap category generated a unique neural network model with specific layer constants and input normalization values. Each neural network model's constants were imported into the Python algorithm, and functionality was developed to apply the appropriate values for layer, bias, and normalization constants depending on the kidnap model against which the amcl data is being evaluated.

The developed Python program delivers the ability to apply MATLAB-trained neural network classification models to the ROS environment. By avoiding reliance on specific MATLAB dependencies in the neural network implementation, the Python program can be used to integrate a neural network trained with any tool with the ROS environment.

To invoke this detection script in a Gazebo simulation, the Python amcl_listener_with_nn.py script can be run after the steps outlined previously (see "Running A Simulation") are run. Similarly, for a live amcl localization process, the script can be initiated after amcl.

Multi-Category Kidnap Detection

The Python script used to detect kidnapping events applies the neural network functions developed for each of the three kidnapping categories to incoming cloud_weight topic messages and thresholds the neural network output to determine whether the cloud_weight data indicates kidnapping events. If kidnapping of a particular category is detected, the script prints a "Kidnapping Detected" message and indicates what kind of kidnapping was detected (Figure 21).

```
Major Kidnapping NN function produced:
[ 2.57863888]
Major Kidnapping NN threshold is:
1.2
Major Kidnapping Detected
Initiating global Localization service.  Redistributing particles across the ma
p.
```

Figure 21: Kidnapping detection message from script

If the Python script has its kidnap correction actions enabled, the global_localization service in the amcl package is initiated from the Python script. Figure 21 outlines the detection method in pseudocode.

For each *cloud_weight* message upon arrival:

//Apply kidnap detection for Major kidnap events

neural_net_output = $f_{Major\ Kidnapping}(make\_into\_histogram(cloud\_weight, \text{bins=10}))$

kidnapping_detected= threshold(neural_net_output, major_kidnapping_model_threshold)

If (kidnapping_detected = True): send_alert("Major Kidnapping Detected")

//Apply kidnap detection for Minor-(X, Y) kidnap events

neural_net_output = $f_{Minor\ X,Y\ Kidnapping}(make\_into\_histogram(cloud\_weight, \text{bins=10}))$

kidnapping_detected= threshold(neural_net_output, minor_xy_kidnapping_model_threshold)

If (kidnapping_detected = True): send_alert("Minor (X, Y) Kidnapping Detected")

//Apply kidnap detection for Minor-Orientation kidnap events

neural_net_output = $f_{Minor\ Orientation\ Kidnapping}(make\_into\_histogram(cloud\_weight, \text{bins=10}))$

kidnapping_detected= threshold(neural_net_output, minor_orientation_kidnapping_model_threshold)

If (kidnapping_detected = True) send_alert("Minor (Orientation) Kidnapping Detected")

Figure 22: Pseudocode for the kidnap detection script

Detecting Prolonged Disturbance Events

A complication that arose from calling the global_localization service immediately upon kidnap detection was that after the first kidnap event, the amcl particle filter data that was signaling kidnap activity gets reinitialized by the global_localization service, which distributes the particle filter poses across the map and normalizes the associated weights (Figure 23). This

makes it impossible to detect a series of consecutive kidnap events that would otherwise indicate prolonged disturbance kidnap conditions.

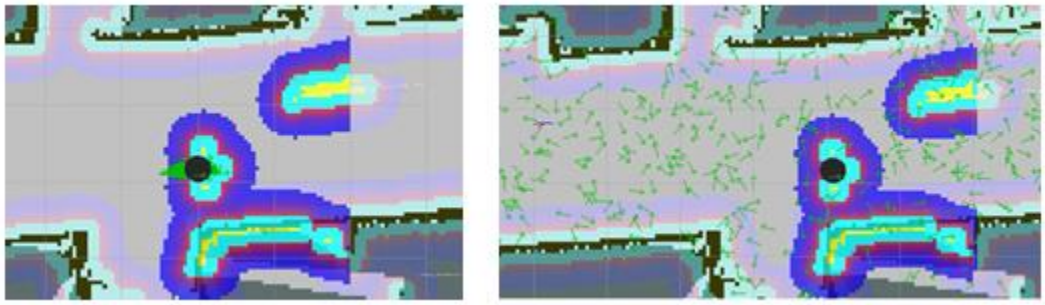


Figure 23: Particles centered around robot current pose (Left). Particles are distributed across the map when the global_localization service is called (Right)

The kidnap detection script was revised to be in a detection-only mode, in which global_localization was not called following a kidnap event. When kidnap events are detected, a counter is updated and a sliding window is applied to determine if there have been 5 kidnap events in the most recent 5 time steps. If kidnapping events have been detected in the past 5 consecutive time steps, the script prints a message to the user (Figure 24).

```
Sliding window currently has the following data for previous timesteps:
Current Timestep (t) has Kidnap data: True
t-1 Timestep has Kidnap data: True
t-2 Timestep has Kidnap data: True
t-3 Timestep has Kidnap data: True
t-4 Timestep has Kidnap data: True
5 consecutive kidnap events were detected. Prolonged disturbance kidnapping is
likely. Consider a new driving strategy.
```

Figure 24: Message resulting when a Prolonged Disturbance event is detected

Figure 25 shows pseudocode for detection-and-correction and the detection-only modes of the kidnap detection script.

```
//If detection-and-correction mode is enabled,

If (kidnapping_detected = True)
        send_alert("Kidnapping Detected")
        call_ROS_AMCL_global_localization_service()


//If detection-only mode is enabled,

If (kidnapping_detected = True)
        send_alert("Minor (Orientation) Kidnapping Detected")
        prolonged_disturbance_detected = check_for_prolonged_disturbance()
        if (prolonged_disturbance_detected = True)
                send_alert("The current and 4 previous time steps have been kidnap events.
                        Consider another driving strategy.")
```

Figure 25: Psuedocode for taking action upon kidnap detection

Delaying Kidnap Detection Initiation

When running the kidnap detection script, it was found that the first three to five time steps during localization exhibit similar cloud_weight messages as those of kidnapping events as the particle filter stabilizes in the environment. To prevent unnecessary global localization activity triggered by the false positives, the kidnap detection script was not initiated in the trials until the robot had been moving for five time steps. In practice, some kind of delay could be implemented in the Python kidnap detection script to automate this wait period.

CHAPTER 5: RESULTS

Testing Models on Live Turtlebot Environment

The Turtlebot robot was provided the data collection code originally developed for the simulated Turtlebot for use with Gazebo. The compatibility between the Gazebo model of the Turtlebot robot with the actual Turtlebot robot made it easy to transfer the code and execute it without issue.

Creating a Map of a Hallway

The gmapping_demo.launch file in the turtlebot_navigation ROS package was used to create a map of a 2.4 meter-by-18 meter school hallway environment (Figure 26).

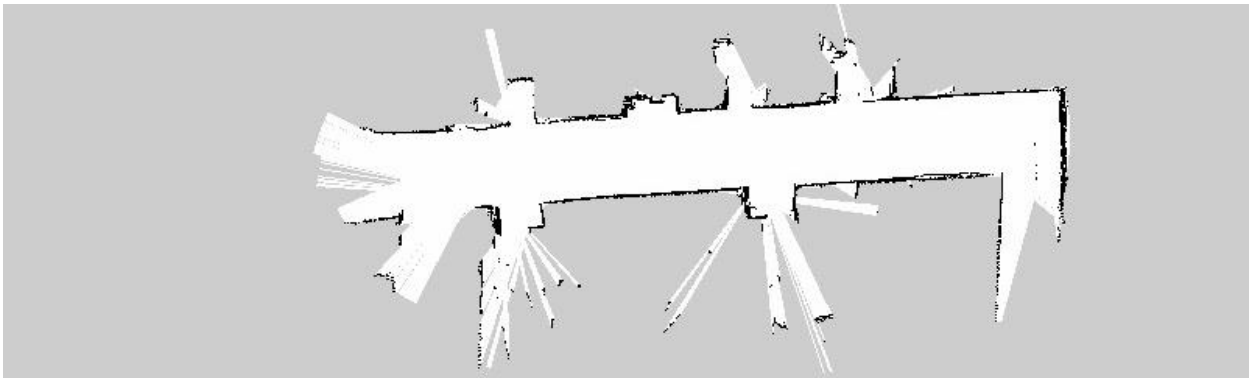


Figure 26: Sensor map created with gmapping of a school hallway

The stretch of hallway, located on the second floor of McDonnell Douglas Hall on the Saint Louis University campus, included recesses for three classroom doors, two restrooms, one supply closet, and one water fountain (Figure 27).

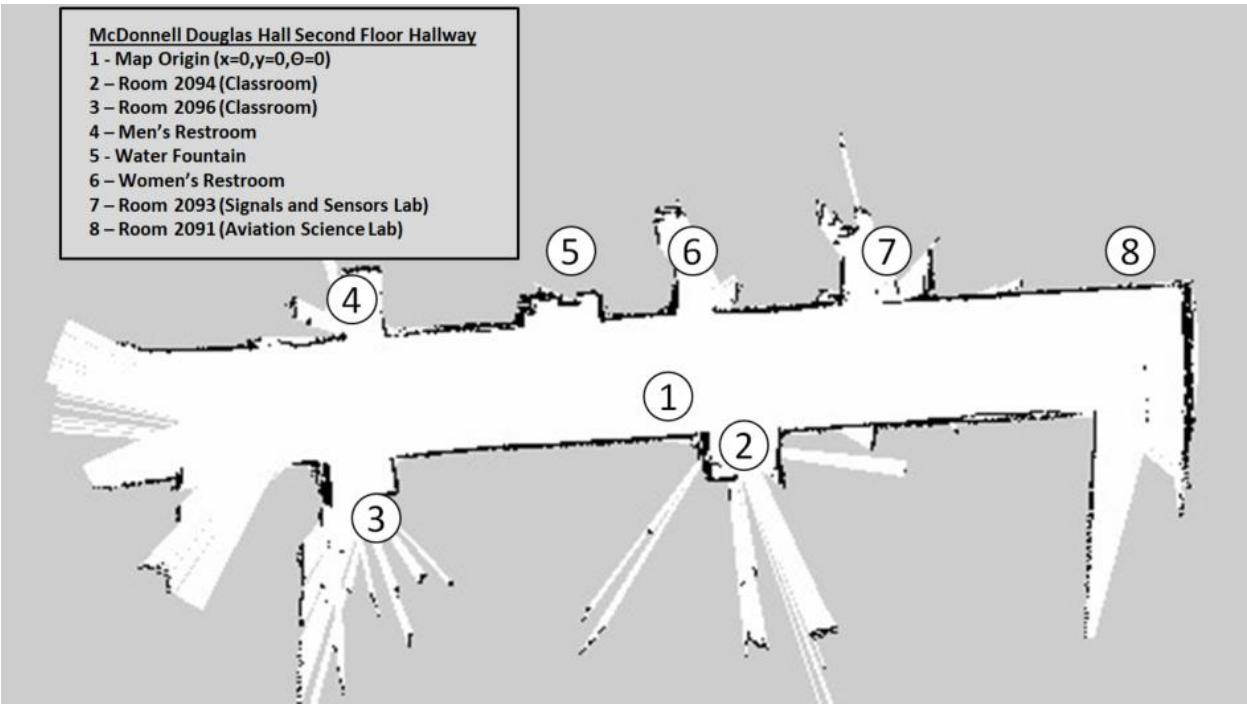


Figure 27: Sensor map with landmarks overlaid

The resulting map was provided to the AMCL algorithm, and the robot was driven around the hallway with the turtlebot_teleop navigation package. During the experiment, the particle filter data was collected for later analysis.

Collecting AMCL Data on the Turtlebot

To use the map with the amcl localization package on the Turtlebot, the start_amcl.sh and start_recording_messages.sh Bash scripts were copied from the virtual machine to the computer associated with the Turtlebot robot. The following commands were executed from a command window in order:

1. *roslaunch turtlebot_bringup minimal.launch*

   This initiates the startup processes on the Turtlebot, such as creating the ROS message handlers and initializing network components.

2. bash start_amcl.sh

Bash statements in this program routed the source directory to the custom AMCL implementation, initiated the amcl_demo.launch file from the turtlebot_navigation package, and provided the file path to the .yaml map to be published by the map_server Publisher.

3. bash start_recording_messages.sh

   This script started the rosbag recording process to save messages published to specified topics for later analysis. The amcl_demo.launch program establishes a connection to the Kinect camera used for laserscan images in localization, so no additional operations are required for initiating the Kinect or other sensors.

4. *roslaunch turtlebot_teleop keyboard_teleop.launch*

   This program provided keyboard mappings that were used to drive the Turtlebot robot.

5. *roslaunch turtlebot_rviz_launchers view_navigation.launch*

   The RViz application enabled visualizations of the environment map provided by map_server, the robot's position in the environment based on amcl, and the amcl particle filter pose estimates.

Challenges Faced When Implementing Code Developed in Simulation onto the Turtlebot

Unanticipated challenges arose when trying to provide the hallway map to the amcl program using the turtlebot_navigation package's amcl_demo.launch launch file. The amcl_demo.launch file accepts a "map_file" argument used to provide the file path for the desired map .yaml file to be published by the map_server Publisher on the /map topic. When the map file path was provided the first time, the map_server process crashed immediately. The error log made note that the map_server was attempting to open the .pgm image associated with the map file from the /tmp file directory on the computer. After researching solutions for

correcting the .pgm map image file path, the easiest option was to copy the .pgm file associated with the specified map into the /tmp directory. This placated the map_server process and the correct map was published to the /map topic. This issue was not encountered in the Gazebo simulation, so its cause is unclear.

Another issue encountered when running amcl with the hallway map on the Turtlebot was an error from amcl_demo.launch that the program was "Waiting on transform from base_footprint to map to become available before running costmap, tf error". The amcl log file showed the following error message: "No laser scan received. Verify that the data is being published on the scan topic." This issue resolved itself after restarting the ROS environment with the turtlebot_bringup package's minimal.launch script. The default amcl_demo.launch file was run successfully, and subsequent custom amcl_demo.launch code was successful. The "rostopic list" command was used to confirm that the custom amcl messages were being published as intended. The start_recording_messages.sh script was used to save the amcl messages into a bag file.

An implementation challenge surfaced while integrating the kidnap detection script into the Turtlebot environment. The kidnap detection script references the custom cloud_weight ROS message topic published by the amcl overlay code base, and this requires a custom message type, named filter_covariance_msg, to be available to the ROS environment. The filter_covariance_msg message type is named such because it was originally developed for transmitting particle filter covariance messages from amcl. The filter_covariance_msg message type was repurposed for the cloud_weight message due to their similar data structures. Although previous steps had verified that the cloud_weight topic was being published on the Turtlebot, the command window in which the kidnap detection script was running received the following error:

"Cannot load message class for filter_covariance_msg." This error is caused by the custom messages not being within the scope of the command window, and has been resolved in the past by calling "source <path to the setup.sh file in the ROS package directory>" to point out the custom messages and ROS packages that are in use. Calling the "source" command did not resolve the issue. Fortunately, a user on the ROS Answer site proposed a solution that identified the issue[39]. The CMakeLists.txt file was missing filter_covariance_msg in its "add_messages" list. Adding the message in CMakeLists and calling catkin_make clean followed by catkin_make to clean and rebuild the dependency chain corrected the dependency definitions. After calling "source" one more time, the kidnap detection method was able to successfully access cloud_weight messages in the ROS environment.

The next issue encountered when implementing the kidnap detection program on the Turtlebot was a version conflict in the matplotlib Python library. The machine on which the code was developed was running a newer version of matplotlib than the Turtlebot's machine, and the Turtlebot machine did not support the histogram function developed on the simulation machine. Attempts to upgrade matplotlib revealed an out-of-date version of the pip version management tool that prevented the upgrade of matplotlib. Attempts to upgrade pip were met with an "InsecurePlatformWarning" error that online research implied was related to the Turtlebot's out-of-date network traffic handling by the operating system. Ultimately, the kidnap detection program was adapted to remove the matplotlib dependency and replace it with numpy classes to perform the same functionality.

The success of the amcl localization on the Turtlebot relies on agreement between the initial amcl position estimate and the robot's true pose. Proper initialization was conducted by starting amcl on the Turtlebot and opening the RViz tool to provide a view into the robot's

AMCL position plotted on the map of the environment. The robot's default pose is (0, 0, 0) in terms of (x, y, Θ), and this location is also the map's origin. The robot's amcl initialization at the map origin was viewed in RViz to determine where to place the robot in the hallway at the beginning of the localization data collection trials. The RViz application includes a Measure tool (Figure 28) that allows a ruler to be drawn from one point on the map to another.

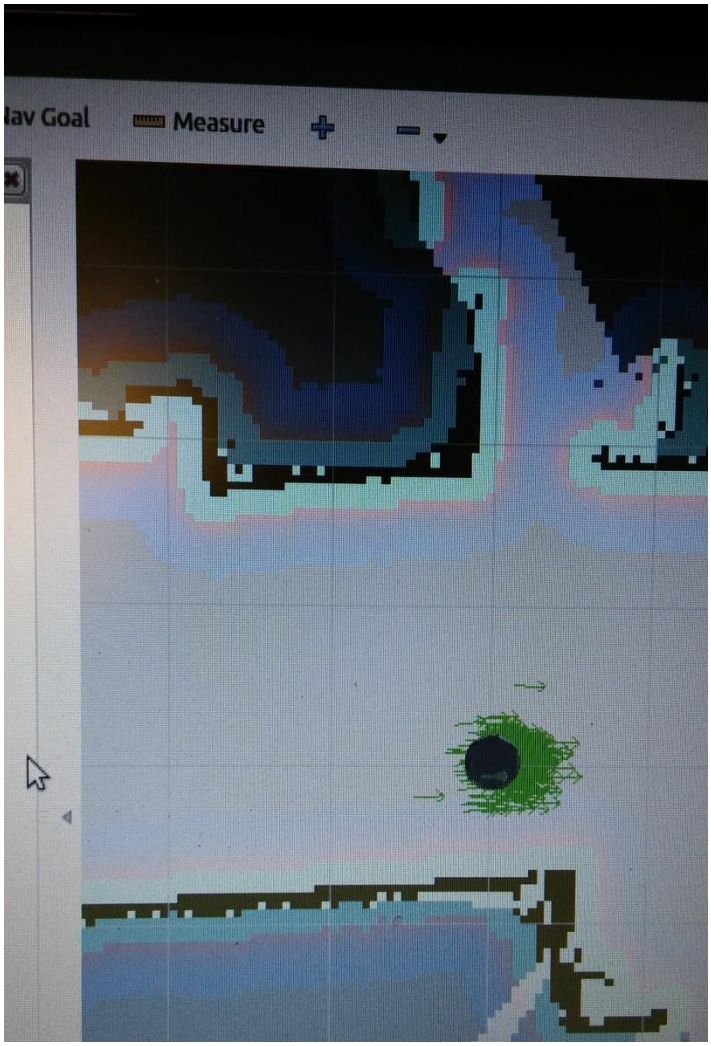


Figure 28: RViz application has a Measure tool that is used to calculate the Turtlebot's location from the wall

Using the Measure tool, the robot's location to an environment landmark – in this case, a particular classroom door - was calculated. In this way, the Turtlebot's starting location in the

hallway was ascertained, and the Turtlebot was placed at the specified measurements near the classroom door for its initial position.

Hallway Benchmark Trial

The accuracy of the AMCL localization service was benchmarked prior to conducting kidnap detection trials. The Turtlebot was driven in a 30-meter figure-eight through the hallway (Figure 29).

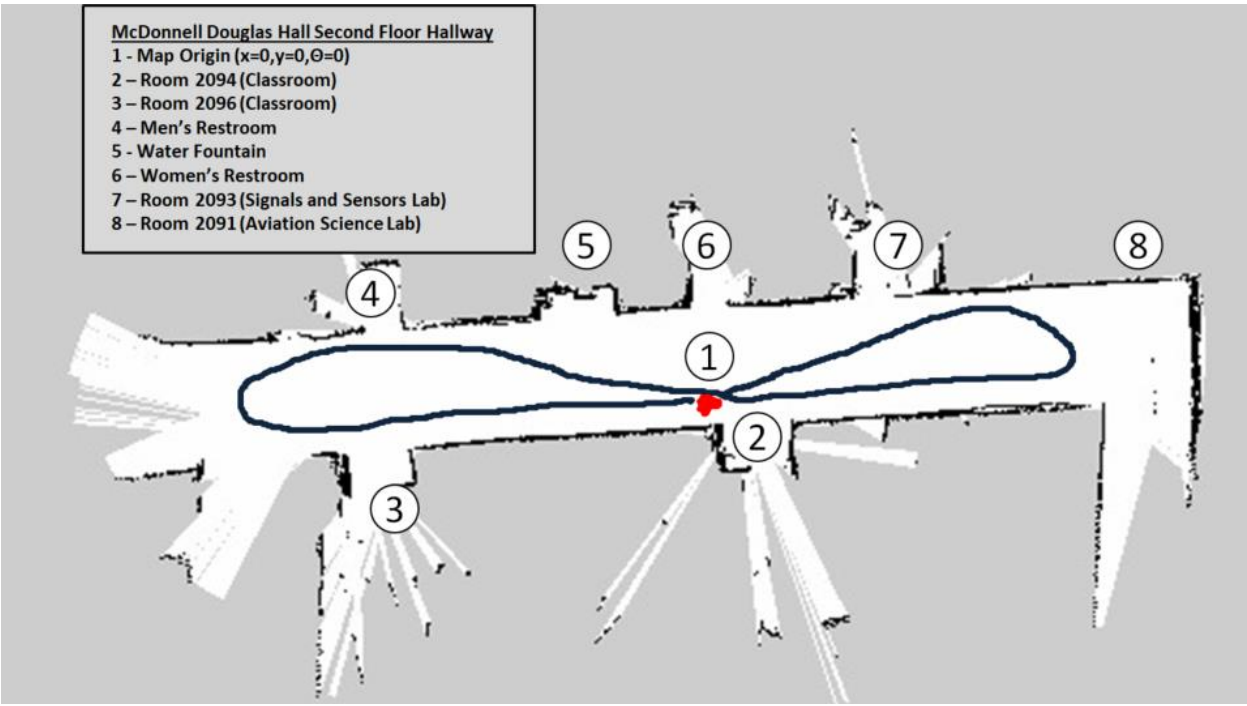


Figure 29: Sensor map with figure-eight benchmark route overlaid

When the robot was returned to the map origin, its true location in (x, y, Θ) coordinates was close to (0, 0, 0). The AMCL estimate of the robot's position in the final timestep of the ROS bag file recording of the amcl_pose topic was (-0.47, 0.08, -0.02). Average yaw localization error in the Gazebo simulation benchmarks was 0.13 radians, so the error present in the final timestep of the hallway yaw is less than the Gazebo simulation average. Average Euclidean error in the (x, y) dimensions in the Gazebo simulation benchmarks was 0.31 meters,

which is lower than the hallway trial's final (x, y) Euclidean distance error of 0.473 meters (Table 5).

Table 5: Benchmarks for AMCL Localization Error

Environment	**Euclidean (x, y) Error (meters)**	**Yaw Error (radians)**
Gazebo Default World (Avg.)	0.31m	0.31 rad
SLU Hallway (Final Timestep)	0.47m	-0.02 rad

Implementing Hallway Data Collection with Turtlebot

To collect data for kidnapping trials, the robot was driven up and down the hallway using the roslaunch turtlebot_teleop keyboard_teleop.launch script. Keyboard_teleop.launch provides keyboard mappings for controlling the Turtlebot's forward, backward, and turning motions. Kidnap events were created by making physical adjustments to the robot's position by picking up the robot and putting it down in a new (*x*, *y*) position or changing the robot's orientation or yaw. The hallway tiles were measured and found to be 0.3 meters squares, and their uniformity provided a natural grid that was used to identify specific "Kidnap Start" and "Kidnap End" locations for comparison against the AMCL pose estimate. The "Kidnap Start" location used in the trials was identified with a sticky note placed on the tile at (2 meters, 0 meters) from the map origin. The Kidnap End locations were other sticky notes on the tile whose placement was a measured distance from the Kidnap Start sticky note and varied by the kidnapping scenario involved in the trial. When the robot was driven over the Start sticky notes, the robot was picked up and put down over a nearby Kidnap End location. For the Major Kidnapping trials, the robot was kidnapped to a Kidnap End sticky note located at (1 meter, 1 meter) from the map origin.

For the Minor-XY Kidnap category, the Kidnap End location was at (2.1 meters, 0.1 meter) from the map origin.

To generate data sets for the Minor-Theta kidnap category, Kidnap End locations were designated in which the robot's pose at the Kidnap End point remained the same, but its orientation was increased by 0.17 radians, 0.34 radians, 0.51 radians, or 1.57 radians, as measured out with a protractor in units of degrees (Figure 30).

Figure 30: A protractor was used to set the Turtlebot's orientation to a Kidnapped pose

In this trial category, an issue was encountered using the Turtlebot that had not occurred in the Gazebo Turtlebot simulations. When the Turtlebot is picked up, the gyroscope on-board the Kobuki base measures the change in yaw and amcl updates its position estimates according to the change in orientation. The update is instantaneous, and even when the robot is picked up and twirled around in multiple full rotations, the amcl estimates are updated to match the robot's orientation changes. This prevented the kidnapping method of picking the robot up and putting it down in the new orientation from achieving its goal of disrupting the agreement of the

exteroceptive and interoceptive sensors to simulate kidnapping. Trials were recorded with the pick-up method anyway, but another trial technique was devised to simulate the kidnap scenario despite this setback. Instead of placing the robot at a yaw value that matches the yaw value provided to the amcl localization algorithm when specifying the robot's initial pose in the environment, the robot was placed at an erroneous yaw for its initial position so that the robot's yaw was off by a certain number of radians. This setup served as a replacement technique for the original kidnapping scenario and allowed data to be collected on the pose uncertainty in the trial.

The prolonged disturbance kidnapping scenarios were simulated by driving the robot in a path in which one wheel drove over a one-meter-long stretch of coarse sand one centimeter deep to create a non-uniform and uneven driving surface (Figure 31).

Figure 31: Sand used for prolonged disturbance simulation

Other options tested and considered included having the robot drive over plastic trash bags, crumpled paper obstacles, a yoga mat, a bedspread, and a canvas tarpaulin (Figure 32). The sand option was ultimately chosen for its ability to dramatically change the driving surface without harming the robot, as other dramatic methods, such as creating ice or an oil slick, might

have. The uneven sand forces the robot to drive according to a driving model that differs from the surface on which the localization map of the environment was created.

Figure 32: Hallway obstacles added to create prolonged disturbance driving conditions

For the hallway trials, a GoPro camera was used to record the video (Figure 33), while the SimpleScreenRecorder application was used on the Ubuntu computer onboard the Turtlebot to record the RViz visualization of amcl activity. These procedures document the trials for later analysis should data results require additional review.

Figure 33: Hallway simulations were recorded by a GoPro camera

amcl Initialization Challenges

While conducting trials with the Turtlebot, it was noticed that although the amcl program was provided with values setting the robot's initial pose in the map to (0 m,0 m, 0 rad), the initial (*x*, *y*) robot pose according to amcl upon start-up and prior to any movement were within 0.4 meters in the positive or negative directions. The initial Θ parameter was off by 0.04 radians in the positive or negative directions. This error was corrected by specifying that the robot's initial pose in the environment was (0.01 m, 0.01 m, 0 rad). Upon examination, it was learned that this same effect was present in the amcl pose data collected from the Gazebo simulations, so it is possible that the amcl code applies some sort of noise to the initial pose settings and reducing the initial pose to 0.01 reduces the level of noise added to the initial pose to negligible amounts.

Hallway Kidnapping Trials

Over 30 trials were run with the Turtlebot in the McDonnell Douglas Hallway environment corresponding to the sensor map of Figure M2 (Table 6).

Table 6: Data Collected with Turtlebot in McDonnell Douglas Hall

Trial Name	Distance Driven	Number of Time Steps	Number of Trials
Benchmark	30 meters	149	1
Major – Kidnapped from (1, 0) to (1, 1)	3 meters	17	5
Minor - Kidnapped from (1, 0, 0) to (1, 0, 0.17)	3 meters	12	2
Minor - Kidnapped from (1, 0, 0) to (1, 0, 0.51)	3 meters	13	3
Minor - Kidnapped from (1, 0, 0) to (1, 0, 1.53)	3 meters	17	1
Minor - Wake-Up Robot - Kidnapped from (0, 0, 0) to (0, 0, 0.17)	1 meters	3	1

Table 6. Continued.			
Trial Name	Distance Driven	Number of Time Steps	Number of Trials
Minor - Wake-Up Robot - Kidnapped from (0, 0, 0) to (0, 0, 0.51)	1 meters	3	1
Minor - Wake-Up Robot - Kidnapped from (0, 0, 0) to (0, 0, 1.53)	1 meters	3	1
Minor - Kidnapped from (1, 0) to (1, 0.1))	3 meters	14	10
Driving With One Wheel on Sand	3 meters	13	5

The Robot Kidnapping Detection method's performance in the trials and various Kidnapping scenarios was analyzed in Table 7.

Table 7: Performance of Robot Kidnapping Detection method on McDonnell Douglas Hall trials

Kidnapping Type	False Negatives	False Positives	Kidnap Events Detected (even if misclassified)	Kidnap Events Misclassified
Major	**28.00%**	**0.00%**	**72.00%**	**52.00%**
Minor	**10.81%**	**11.76%**	**89.19%**	**8.11%**
Immobilized	**0.00%**	**n/a**	**100%**	**0.00%**
Minor Prolonged Disturbance - Sand	**38.46%**	**11.76%**	**61.54%**	**3.85%**
Wake-Up Robot (Minor Error in Initial Theta)	**26.62%**	**11.76%**	**73.33%**	**20.00%**

Notable performance statistics include the 11.76% of time steps during the benchmark driving route being detected as Minor Kidnapping events. It is possible that naturally-occurring minor wheel slips were being detected in these time steps, or that certain points in the hallway were sparse on distinguishing features and caused an increased pose uncertainty. The Major

Kidnapping trials detected Kidnapping in 72% of trials, but misclassified 52% of post-Kidnapping time steps as Minor Kidnapping events. This can be explained by the particle filter's behavior when encountering pose uncertainty. When pose uncertainty exists, the AMCL algorithm increases the number of random particles in the filter incrementally at each time step. In the first time steps following a Major Kidnapping event, the particle filter state resembles the state following a Minor Kidnapping. As time passes and more random particles are added to the particle filter, the particle weight imbalance indicative of Major Kidnapping become pronounced enough for the Major Kidnapping model to detect the event. In this work, global localization was initiated when a Minor Kidnapping event or a Major Kidnapping event were detected, so the end result was identical whether the Major Kidnapping event was classified as Major or Minor.

The trials in which one of the robot's wheels was rolled over sand while the other wheel remained on the tile simulated the Prolonged Disturbance Kidnapping category, in which sequential Minor Kidnapping events lead to pose uncertainty. The kidnapping events of the Sand trials were detected 61% of the time. In each trial, the robot drove over the sand for five or six time steps before the Kidnap event was detected. It is thought that the sand was a good approximation of wheel slip, but that the wheel slip was so minor that it took several time steps for the pose uncertainty to set off the detection model.

The above Major and Minor trials produced datasets that could also be examined to analyze the models' performance under the other Kidnapping categories of Immobilized and Wake-Up Robot Kidnapping. When the robot became stuck on the wall in three of the Minor trials, the resulting data showed that 100% of Immobilization time steps were recognized as Kidnapping events by the Major and Minor Kidnap Detection models. The trials implemented as a work-around for the gyroscope's detection of changes in the robot's orientation served as

Wake-Up Robot simulations, in which the robot's knowledge of its initial position is unavailable, or in this case, corrupted. The Kidnap Detection models identified 73% of time steps in these trials as Minor or Major Kidnapping events.

It is clear that the Kidnap Detection in the McDonnell Douglas Hall trials performed worse than the Kidnap Detection on the Gazebo trials. Major Kidnapping events were detected with a 4.78% False Negative rate in the Gazebo trials, and the McDonnell Douglas trials resulted in a 28% False Negative rate. The difference can be attributed to the difference in environments of the trials. The Gazebo environment included unique landmarks that could be used by amcl to strengthen pose certainty. The McDonnell Douglas trials were conducted in a hallway with few distinct features, which challenges the scan-matching component of AMCL as it matches the robot's sensor view with a single pose on the map.

CHAPTER 6: DISCUSSION

Method Performance

Qualitative Comparison

The proposed Kidnap Detection method satisfies the qualitative properties that are used to evaluate state-of-the-art Kidnap Detection schemes. The Kidnap Detection scheme not only detects the Kidnapped Robot Problem, but it differentiates between the main categories of Robot Kidnapping and advises the user of the Kidnapping type. The proposed Kidnap Detection scheme is easy to transfer and use on robots. The computation efforts of the detection script's addition, subtraction, comparator, and simplistic matrix multiplication operations do not cut into the robot's battery life or computational resources in any significant way. The laptop associated with the Turtlebot robot platform was monitored while the Kidnap Detection scheme ran concurrently with localization activity, and the process added 0% CPU load and consumed 200KB of memory. This is a very small process compared to other ROS processes such as amcl, which consumes 9.5MB of memory.

Another strength is the proposed method's ability to operate regardless of sensor type. The proposed Kidnap Detection system does not use sensor data directly in its calculations, making the method sensor-independent. Working with only the particle filter weights enables this Kidnap Detection method to ignore the particle filter details that are specific to the map and avoids the extra step of having to normalize the particle filter poses to account for different map origins. One of the weaknesses that Campbell et al.[11] mention about their own Kidnap Detection method is that classifiers like SVMs or neural networks use thresholds tuned to work best for the specific sensor data produced by the localization environment. Changes to the sensors or the environment require refining the thresholds used in the classifier. In the proposed method, thresholds are tuned based on uncertainty in the localization algorithm, and do not

require retuning for changes to the sensors or environment. However, the difference in model performance between Gazebo and the McDonnell Douglas Hallway show that localization environments with sparse features may cause more uncertainty in the localization process than environments with more numerous and distinct features. More uncertainty in the environment was shown to cause the Kidnapping Detection models to register additional false positives, particularly in the Minor Kidnapping model.

While Bukhori et al.[12] discount the Kidnap Detection methods devised by Zhang et al.[13] and Yi et al.[16] because the particle filter's confidence weights may be misleading about the accuracy of the particle filter pose estimate, the proposed Kidnap Detection method bases its classifier on the code behind the localization algorithm that has been shown to cause specific pose weight distributions as a result of Robot Kidnapping. Monte Carlo Localization is one of the most popular localization algorithms used in mobile robotics, and any Adaptive variety of MCL has similar properties to the weight distribution identified in this work that can be exploited for Kidnap Detection. With the proposed method, robots running amcl can detect and correct Kidnapping events to mitigate the risks of operating with an incorrect pose.

The Kidnap Detection method's limitations include its tight coupling to the ROS amcl localization implementation. The proposed Kidnap Detection method does not satisfy the Localization Method Independent criteria of state-of-the-art Kidnap Detection schemes. Prerequisites for using the system are that the robot must be using the ROS operating system and the amcl localization algorithm. Another drawback of the method is that the amcl overlay codebase will need to be kept in sync with the latest amcl versions as they are released. Otherwise, users would have to make the choice between the latest amcl functionality of Robot Kidnapping Detection. Campbell et al.[11] mention that large training sets were necessary for

their SVM Robot Kidnapping classifier, and list this fact as a draw-back for their method. Artificial Neural Networks as used in this work also require significant training data, but this data was obtained using the Gazebo simulation and it was relatively convenient to generate the necessary data. Despite this method's limitations, it is a viable Kidnap Detection option for the ROS amcl package for any robot and any sensor configuration. This method can be easily adapted for other localization methods derived from the Adaptive Monte Carlo Localization algorithm.

Quantitative Comparison

Comparing the proposed method's Speed of Detection and False Positive Rate against those of other Kidnap Detection methods provides context for the method's performance. When comparing the performance of Kidnap Detection methods, it should be remembered that there is little standardization of robot environments, and the methods being compared differ substantially in their implementation. This is a research area that would benefit from standardized benchmark tests.

Table K1 summarizes the proposed method's performance compared to the metrics used in other works. In the Speed of Detection category, the proposed method falls into the middle of the range of Kidnaping Detection algorithms' performance. The proposed method can detect a Major Kidnapping event in an average 2.74 seconds. This is faster than the Lee et al. Kidnap Detection method[1] that has a Speed of Detection rate of 68 seconds, but slower than the method for Wheel Slip Detection proposed by Ward et al.[4], with a detection rate of 0.4 seconds or Tian et al.'s 13μs detection rate[10]. The limiting factor of the proposed Kidnap Detection method is the frequency at which the ROS amcl algorithm publishes pose updates. amcl publishes pose updates and the messages added in this work at a rate of once per second or more frequently if

the robot is moving. The proposed Kidnap Detection method's Speed of Detection rates are tethered to the amcl update rates. Although amcl update rates can be customized to perform more updates more frequently than the default once per second, it is unlikely that amcl update rates would be set such that the proposed Kidnap Detection method could achieve Speed of Detection rates on the same level of Tian et al.'s $13\mu s$ detection rate.

Some papers list the Kidnap Detection's ability to correctly detect Kidnapping events as an overall "Accuracy" score, while others put results in terms of False Negatives and False Positives. Results in this work were evaluated in terms of Kidnap Event Detection and Kidnap Event Classification because it was found that Kidnap events were frequently detected first by Kidnap Type detection models other than the Kidnap Type that actually occurred. For example, a Major Kidnapping Event was first detected by the Minor Kidnapping detection model, and the Major Kidnapping detection model detected the Major Kidnapping event in subsequent time steps. When comparing the proposed method to other studies, both the Gazebo trial results and the McDonnell Douglas Hall trial results are included. The Gazebo trials represent an environment with well-defined features, and the McDonnell Douglas Hall trials represent the Detection method's results in a sparsely-featured environment with much uncertainty. Campbell et al.[11] have an enviable False Negative rate for Major Kidnapping events of 0.3%, which outperforms the proposed solution's rate of 4.78% in the Gazebo trials and 28% in the McDonnell Douglas Hall trials. The proposed method's False Negatives rate for Wheel Slip Kidnapping events is 1.45% in the Gazebo trials and 10.81% for the McDonnell Douglas Hall trials. Even in the challenging hallway environment, the proposed method out-performs Campbell et al.'s 18% False Negatives rate for the same category. The proposed method's rates

are higher than the method proposed by Ward et al.[4], which achieves a 0% False Negatives rate for detecting Wheel Slip events.

Distributing This Work

The executables necessary for replicating this work on other robots or simulated robots have been added to GitHub[46], a popular source control website, and are available for free download by the public. To make it easy to use the code, the GitHub repository includes documentation and build instructions. Other source code developed for this research is also available on the GitHub, such as the data visualization MATLAB scripts, the Python neural network code, and the R code used for combining data files. The code is included in Appendix L.

Notices of user-contributed ROS enhancements are usually published to the ROS Discourse website on the General forum channel[47]. The following notice will be published to ROS Discourse upon project completion.

ANNOUNCEMENT: A user-contributed Kidnap Detection scheme for the ROS amcl localization package is available for free download in the linked GitHub repository. Kidnap Detection is performed with a stand-alone Python script that checks the weight distribution of the amcl particle filter and gauges pose uncertainty. The method differentiates between Major and Minor Kidnapping, and alerts the user to sequential Kidnapping events, caused by scenarios such as a Flat Tire or changes in driving terrain. When Robot Kidnapping is detected, the amcl global_localization service is initiated to reinitialize the particle filter to the robot's current pose. This Kidnapping Detection module is available for use on any robot using amcl for localization, and no robot-specific tuning is necessary. Kidnapping Events are detected with 72% accuracy within 2 seconds of the Kidnapping Event.

Other Applications of This Work

The tools developed in this research can be applied in other robotics applications. The analysis of the particle filter developed in this work indicates that it may be possible to obtain actionable information from the behavior of the particle filters used in other processes. The method of analyzing the localization algorithm's internal state during Kidnapping Conditions could benefit the development of Kidnap Detection schemes for other localization algorithms besides amcl. This work developed the concept of performing Kidnap Detection by publishing localization data to ROS topics so that a separate ROS node can consume the localization data and act on the information. This concept could be applied in other Kidnap Detection methods. The novel approach for simulating kidnap events with the Gazebo simulator developed in this work can be extended beyond mobile robots and robot localization to other robot platforms and fault simulation scenarios, such as motor failure in a manipulator arm link.

Evaluation of the ROS and Gazebo Tools

The integration between ROS and Gazebo was impressive. The work in this research used only a fraction of the total capability offered by the tools. A prerequisite for this work should have been that the researcher was already very comfortable with the ROS environment. The learning curve for ROS is steep. The user guides and books available for ROS and Gazebo are useful, but the surface-level use-cases in the examples included in these resources did not provide the depth necessary to be useful for the custom functionality developed for this work. Besides tracking down answers for the "is that even possible in ROS/Gazebo?" questions, unexpected implementation issues occurred regularly in this research. Building the ROS packages with the catkin build tool could have been a straightforward process, but there were many environment linking issues associated with adding custom ROS message files to the ROS

package. Similarly, much time was spent creating the Gazebo plugin and using catkin to compile the files and link dependencies.

The software development debugging tools used or not used for this research added complexity. Text editors were used to create and edit the C++ and Python code bases, instead of using an Integrated Development Environment (IDE) tool with a graphical user interface. Using an IDE would have facilitated the use of code breakpoints and reduced the time spent debugging. ROS packages can be integrated with the popular Eclipse editor for C++, but doing so requires special compilation options in catkin. Late in this research when adding the code to an IDE was considered as a resource for debugging the amcl internal data structures, it was decided that the risk of breaking the packages by recompiling the code base with the catkin options necessary for the Eclipse IDE was too great. Another tool-related challenge was working without the ROS rqt application until late in the project. ROS rqt is an IDE for working with ROS data files, including bag files. The Logging plugin in rqt visualizes the message progression of a bag file's topics and lets the user step through each time step to view the available messages (Figure 34). ROS system logs are also available for viewing in this console, and it was the only way found in this work to view log topics such as ROS_DEBUG. It is important to note for implementation that when using rqt when other ROS processes are not running, the "Could not find ROS master" error can be fixed by opening a new terminal and running the "roscore" command to instantiate the ROS environment. Once discovered, the rqt tool was instrumental in understanding the amcl package.

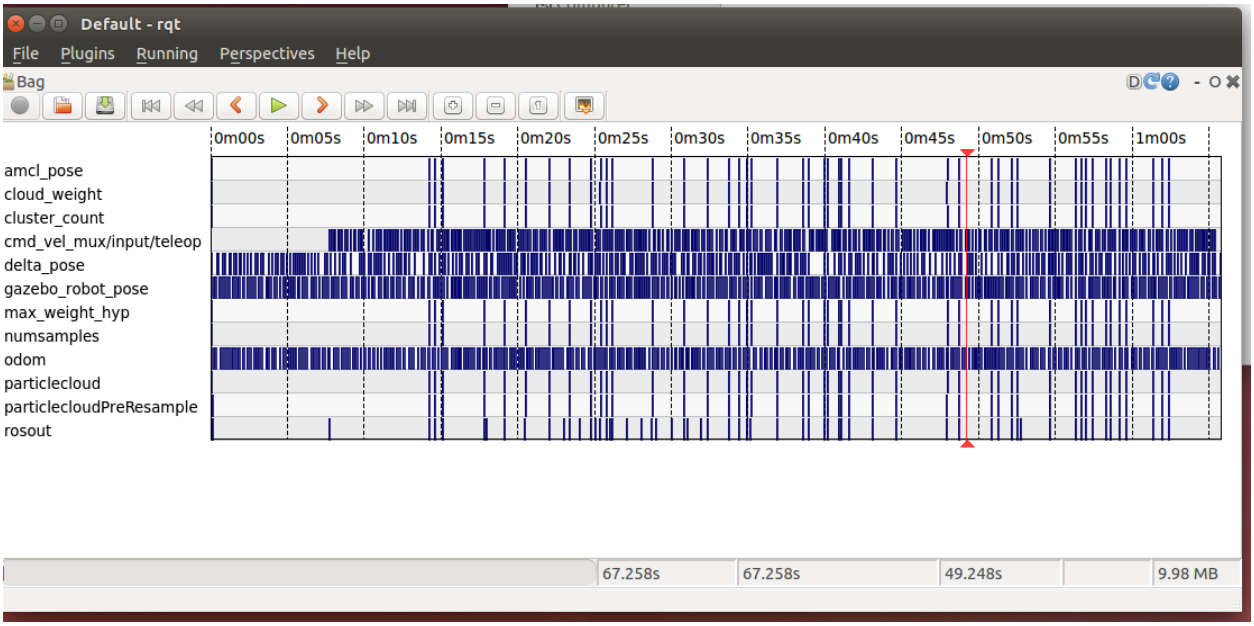


Figure 34: rqt view of a bag file

The ROS Answers community was a helpful resource and users often responded within a few hours to the forum questions related to this research. Such an active and friendly user community indicates that ROS is a main player in the robotics research space. Many of the environment bugs and limitations encountered in this research were reported or discussed in forum posts on the ROS Answers and the Gazebo community sites. This was beneficial to confirm that nothing more could be done on some of the issues, but it was also frustrating to see that the dates on the forum posts indicate that these have been known issues for years and continue to be unresolved in current ROS versions. The inference to be made from this is that there are many more ROS users than ROS contributors, and if ROS users helped out by making small patches to the code when such errors or limitations are found, the ROS codebase would be healthier and more reliable as a whole.

Future Work

Detecting The Kidnapped Robot Problem

Data collected in this work indicates that it is likely possible that the Off-Map Kidnapping scenario can be classified with the same framework as the Kidnapping categories tested in this work. In the Off-Map Kidnapping scenario, the robot is moved to a location not included in the environment map. Particle filter data collected following an Off-Map Kidnapping event showed a uniform distribution representing the uncertainty applied to all of the particle filter poses (Figure 35).

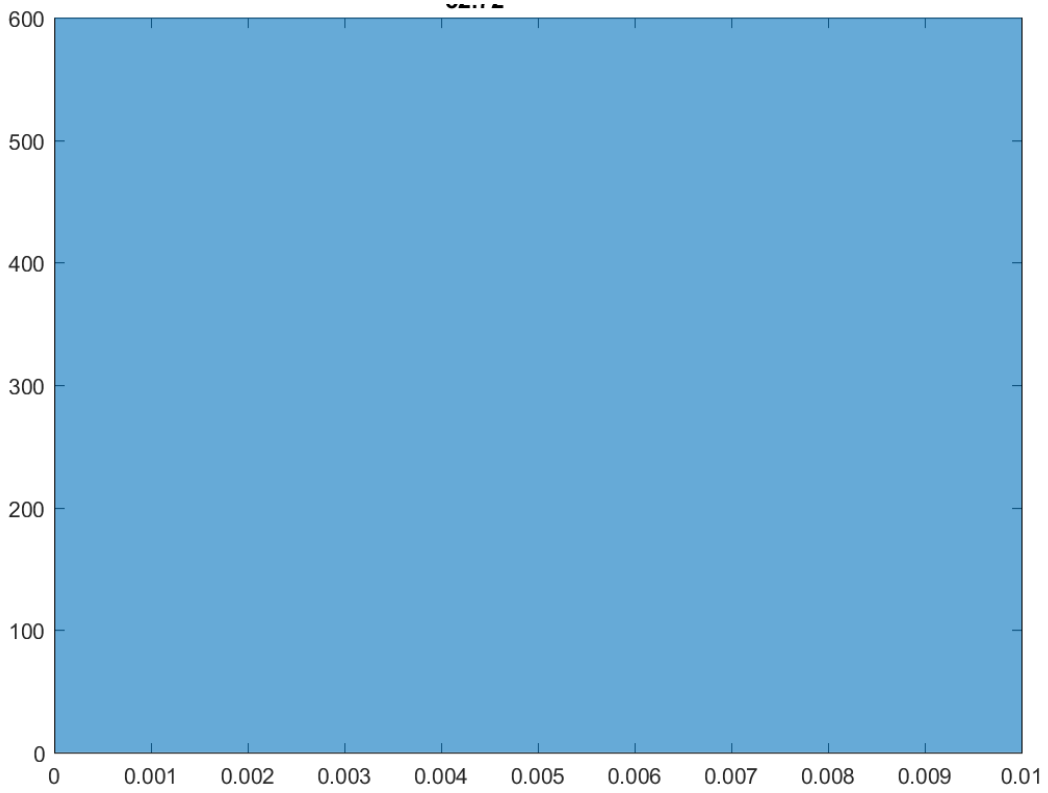


Figure 35: The histogram data produced when the robot is kidnapped to a location not included in the amcl map in an Off-Map Kidnapping event

Adding functionality to detect Off-Map Robot Kidnapping events would make this system more robust. Another area of interest for future work is to evaluate other robotics

platforms' implementations of the AMCL algorithm to confirm that the methods developed in this work can be applied to any robot running AMCL.

Reading the principles of the AMCL algorithm from Probabilistic Robotics [5] indicates that the AMCL short-term and long-term measurement likelihood averages in the Sensor Update phase of AMCL shows promise as an indicator of robot kidnapping. The short-term and long-term averages are already being used in AMCL to determine the quantity and frequency of adding random particle poses to the particle filter to combat pose uncertainty and prevent Robot Kidnapping. Page 259 of Probabilistic Robotics gives a detailed description of how the fast and slow averages are engaged in Robot Kidnapping events. Future studies could use the tools proposed in this work to log the short-term and long-term measurement likelihood averages from amcl in ROS and analyze their performance as an indicator of Robot Kidnapping.

Simulating Robot Fault

The most exciting result of this work is the Robot Kidnapping simulation environment developed in the Gazebo simulator and the novel approach for injecting fault into the robot model's motion as the robot model is controlled by other ROS processes. It would be interesting to generalize the capabilities developed in this research into a package that lets its fault simulation methods be applied to any robot model in Gazebo, such as a manipulator arm link. Once the simulation environment has been enhanced to simulate fault in other robot models, developing a user interface for configuring the fault severity and other properties would make it easy for users to simulate robot fault in Gazebo.

Conclusion

This thesis research began with an abstract concept of The Kidnapped Robot Problem. Five programming languages and two operating systems were used in this work. Through research and experimentation, novel processes were developed for simulating the Kidnapped

Robot Problem in Gazebo. The component parts and resulting data of the Adaptive Monte Carlo Localization algorithm were analyzed, and a novel classifier for identifying Robot Kidnapping events was proposed. The work ultimately showed that the internal processes of probabilistic algorithms can be modeled for Kidnap Detection. The developed Kidnap Detection scheme is light-weight for easy incorporation of Kidnap Detection into ROS robots to improve robot localization operations.

Appendix A: Challenges Using Custom Gazebo World and amcl Map

Originally, a world file was developed in Gazebo to model the 255-square meter dimensions of Room 2084 in McDonnell Douglas Hall on the Saint Louis University campus (Figure A1). A sensor map of the Gazebo environment for use with amcl was created with the gmapping package [40] from the turtlebot_navigation[22] package in ROS (Figure A2). Creating an adequate sensor map with gmapping for such a large space required several attempts at driving the same loop within the simulated classroom. An example of the poor-quality map created in initial attempts with gmapping is shown in Figure A2. Mapping results were improved (Figure A3) by driving the robot around the room several times and remaining close to the walls in the simulation room. The Gimp image editor was used to correct crooked corners in the sensor map, as suggested by [28]. Coordinating the map origin of the global_costmap within the gmapping sensor map with the origin of the Gazebo classroom world was unsuccessful due to a mismatch in the map's yaw or orientation with that of the Gazebo world. This caused the laserscan component of the map data to be misaligned with the Gazebo classroom world's objects (Figure A4). A forum post was issued to the ROS Answers online community website, but users were unable to generate a workaround [41]. Online research found that the ROS global_costmap class does not accept changes to its orientation[42], and that a fix would touch several layers of the ROS architecture, making it a complex patch[43]. This problem continues to be reported as an issue in the ROS community as recently as February 2018 and is considered a "known issue" with Gazebo and amcl[44]. After this knowledge was uncovered, it was determined that using the demo Gazebo world and its matching amcl map would result in a viable environment in which to perform data collection.

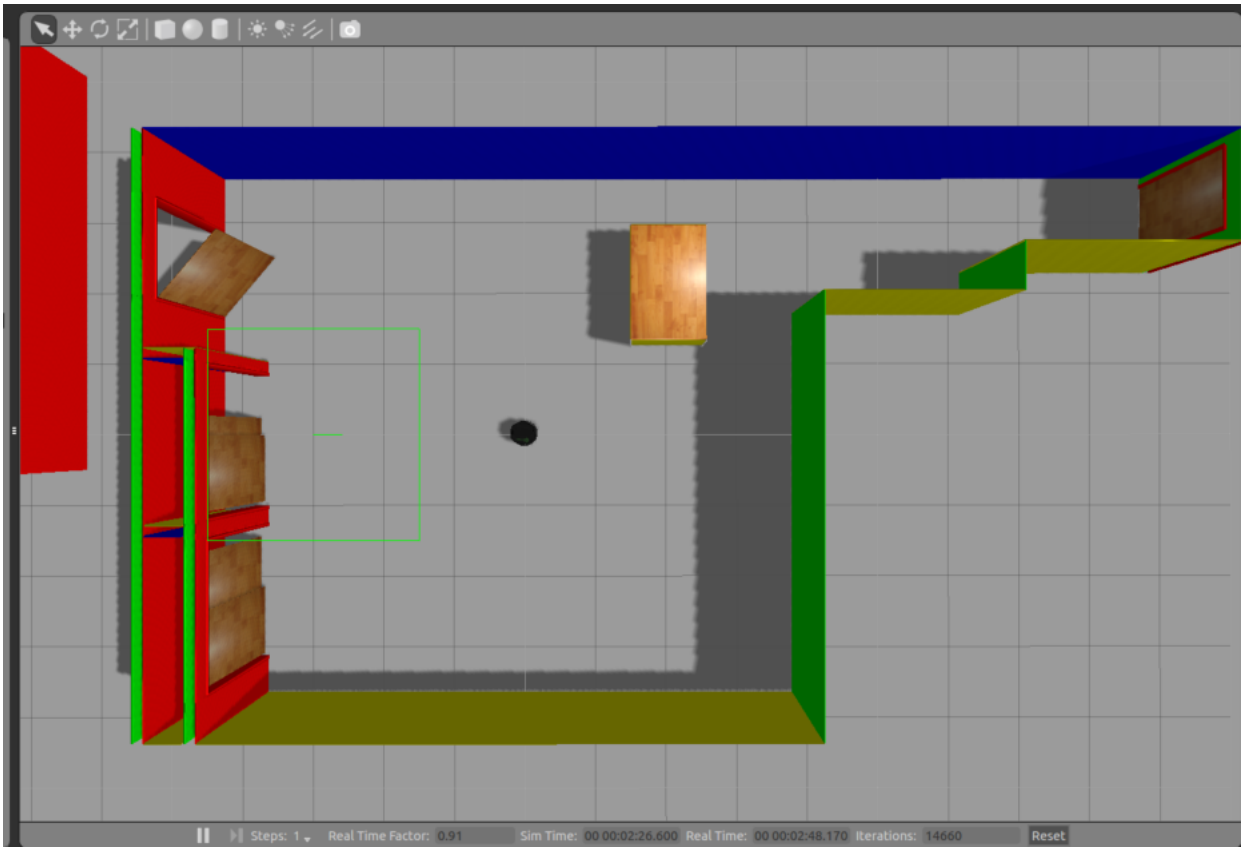


Figure A1: Gazebo simulation world developed based on dimensions of Room 2084 of McDonnell Douglas Hall on the Saint Louis University campus

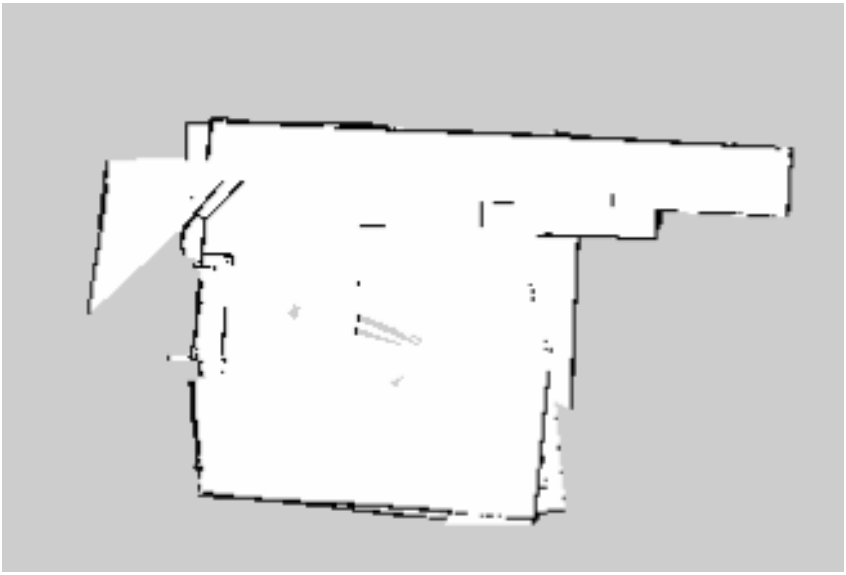


Figure A2: Early attempt at using gmapping to create a sensor map of the classroom

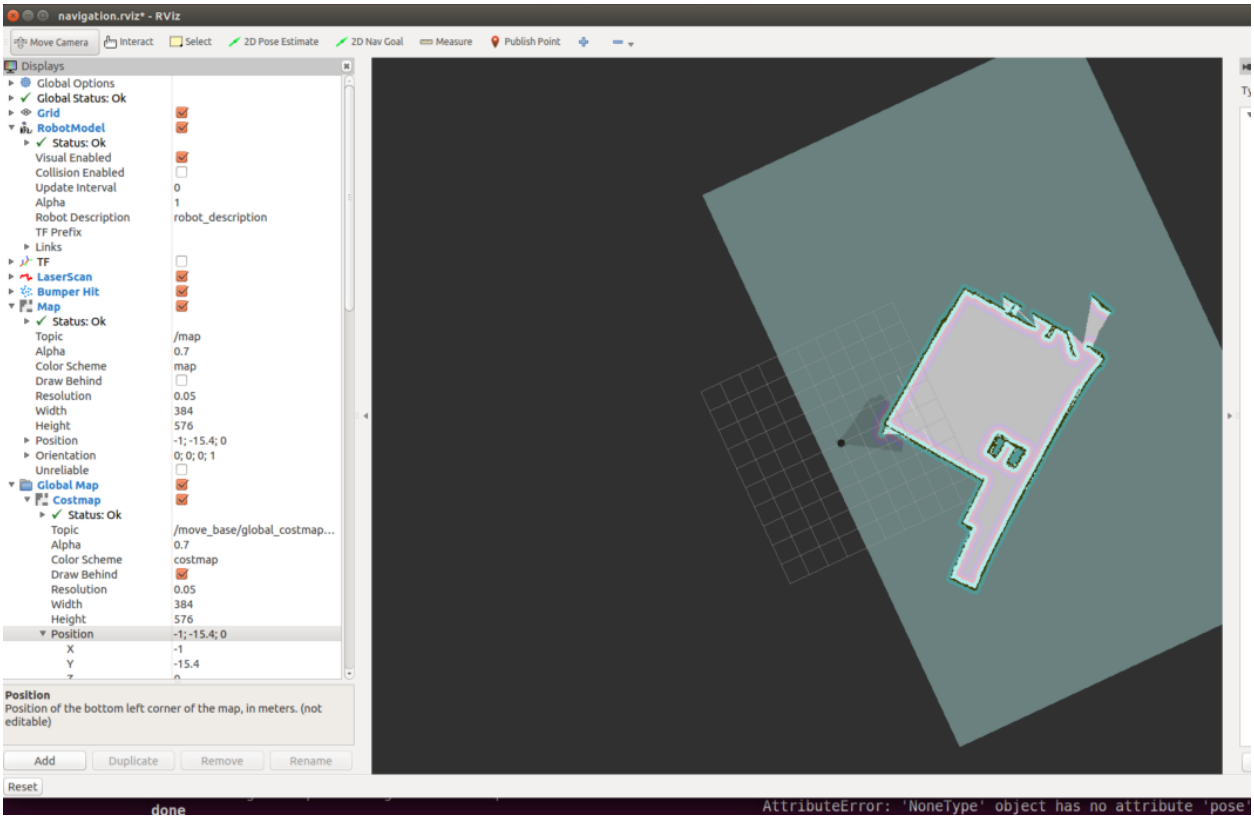


Figure A3: gmapping sensor map of the Gazebo world as visualized with the RViz tool

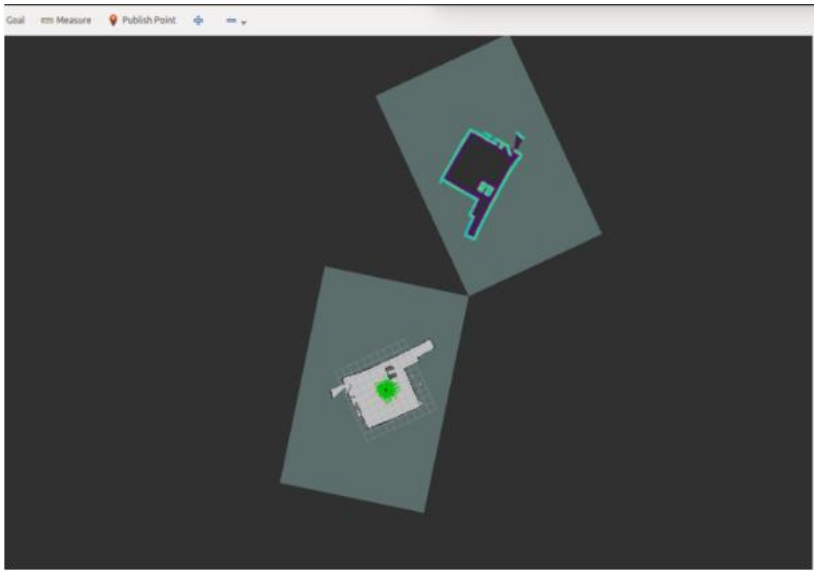


Figure A4: Misaligned sensor map and Gazebo world

Appendix B

Table B1: Topics Recorded in Robot Kidnapping Trials

Topics Recorded in Robot Kidnapping Trials with *rosbag*		
Topic Name	**Description**	**Processes Publishing to Topic**
/rosout	Standard ROS topic used for logging	The Gazebo simulation plugin uses *rosout* to log the kidnapping event timestamps, the robot model's original Gazebo pose, and robot model's current Gazebo pose immediately following the kidnapping event.
/amcl_pose	The (x, y, Θ) estimate of the robot's current pose as calculated by the amcl algorithm	The official amcl package in ROS publishes to /amcl_pose
/particlecloud	An array of every particle in the current amcl particle set, where each particle stores a pose in terms of x, y, and Θ.	The amcl overlay package publishes messages on this topic. This message is published AFTER the particle weights are normalized following the Sensor Update step of the AMCL algorithm.
/particlecloudPreResample	An array of (x, y, Θ) poses for every particle in the current AMCL particle set.	The amcl overlay package publishes messages on this topic. This message is published BEFORE the particle weights are normalized in the Sensor Update step of the AMCL algorithm.
/gazebo_robot_pose	Logs the robot's current position in the Gazebo simulation in terms of x, y, and Θ.	The Gazebo plugin developed in this work publishes messages on this topic.
/cluster_count	Number of clusters in the current amcl particle filter	The amcl overlay package publishes this message
/delta_pose	A comparison of the current and previous amcl pose estimates. This topic sends a pose of (x, y, Θ), calculated as $(x_t - x_{t-1}, y_t - y_{t-1}, \Theta_t - \Theta_{t-1})$ for time t.	The amcl overlay package calculates the values and publishes messages on this topic.

Table B1. Continued.		
Topic Name	**Description**	**Processes Publishing to Topic**
/cmd_vel_mux/input_teleop	Velocity commands sent to the Turtlebot robot by the turtlebot_teleop driving program	The turtlebot_teleop service publishes messages to this topic.
/odom	Message that tracks the robot's accumulated odometry and publishes a transform between the "odom" coordinate frame and the "base_link" coordinate frame, or the coordinate frame representing the robot body[20].	The turtlebot_teleop service publishes messages to this topic.
/cloud_weight	An array of floating point values corresponding to the poses of /particlecloudPreResample and represent the weight or confidence associated with each pose.	The amcl overlay package publishes messages to this topic. This data exposes the weight values that will be used to select the most likely pose estimate in the particle filter for the robot's current pose.
/numsamples	The number of samples currently present in the amcl particle filter.	The amcl overlay package publishes messages to this topic.
/max_weight_hyp	The (x, y, Θ) pose hypothesis with the greatest weight or confidence in the amcl particle filter. It was found that this message is unnecessary because the /max_weight_hyp messages match the already-published /amcl_pose message.	The amcl overlay package calculates the pose and publishes messages to this topic.
/filter_covariance	An array representing the covariance or uncertainty for the amcl particle filter.	The amcl overlay package publishes messages to this topic.

Appendix C

Messages that were created for publishing amcl variables to ROS topics:

Table C8: ROS Messages developed for publishing amcl variables to ROS topics

Message Type Name	Property Name	Property Data Value
amcl_hyp_t_msg	weight	float32
	pf_pose_mean	pf_vector_t_msg
	pf_pose_cov	pf_matrix_t_msg
cloud_weight_msg	cloud_weights	float32[]
cluster_count_msg	cluster_count	int32
delta_pose_msg	v1	float32
	v2	float32
	v3	float32
filter_covariance_msg	cov	float32[]
max_weight_hyp_msg	max_weight	float32
	max_weight_hyp	int32
num_samples_msg	numsamples	int32
num_samples_msg2	numsamples	int32
pf_matrix_t_msg	row1	float32[]
	row2	float32[]
	row3	float32[]
pf_vector_t_msg	v	float32[]
pose_hyps_msg	hyps	amcl_hyp_t_msg[]
test_array_msg	numsamples	numsamples_msg[]
test_msg	testNumber	int32

Table C2: ROS topics publishing amcl variables during localization

Topic Name	Data Description	Comments
/cloud_weight	An array of floating point values corresponding to the poses of /particlecloudPreResample and represent the weight or confidence associated with each pose.	The amcl overlay package publishes messages to this topic. This data exposes the weight values that will be used to select the most likely pose estimate in the particle filter for the robot's current pose.
/numsamples	The number of samples currently present in the amcl particle filter.	The amcl overlay package publishes messages to this topic.
/max_weight_hyp	The (x, y, Θ) pose hypothesis with the greatest weight or confidence in the amcl particle filter. It was found that this message is unnecessary because the /max_weight_hyp messages match the already-published /amcl_pose message.	The amcl overlay package calculates the pose and publishes messages to this topic.
/filter_covariance	An array representing the covariance or uncertainty for the amcl particle filter.	The amcl overlay package publishes messages to this topic.
/cluster_count	Number of clusters in the current amcl particle filter	The amcl overlay package publishes this message
/delta_pose	A comparison of the current and previous amcl pose estimates. This topic sends a pose of (x, y, Θ), calculated as $(x_t - x_{t-1}, y_t - y_{t-1}, \Theta_t - \Theta_{t-1})$ for time t.	The amcl overlay package calculates the values and publishes messages on this topic.
/particlecloud	An array of every particle in the current amcl particle set, where each particle stores a pose in terms of x, y, and Θ.	The amcl overlay package publishes messages on this topic. This message is published AFTER the particle weights are normalized following the Sensor Update step of the AMCL algorithm.
/particlecloudPreResample	An array of (x, y, Θ) poses for every particle in the current AMCL particle set.	The amcl overlay package publishes messages on this topic. This message is published BEFORE the particle weights are normalized in the Sensor Update step of the AMCL algorithm.

Appendix D:

start_recording_messages.sh

```
#!/bin/bash
clear
if [ $# -eq 0 ]
then
echo "No Name for the Bag was Supplied."
exit
fi
echo "Recording ROS topics to ..."
echo $1
rosbag record -O $1 /cluster_count /delta_pose /filter_covariance /max_weight_hyp
/numsamples /particlecloud /test_pub /amcl_pose /cloud_weight /particlecloudPreResample
/odom /cmd_vel_mux/input/teleop /gazebo_robot_pose /rosout
```

start_gazebo.sh

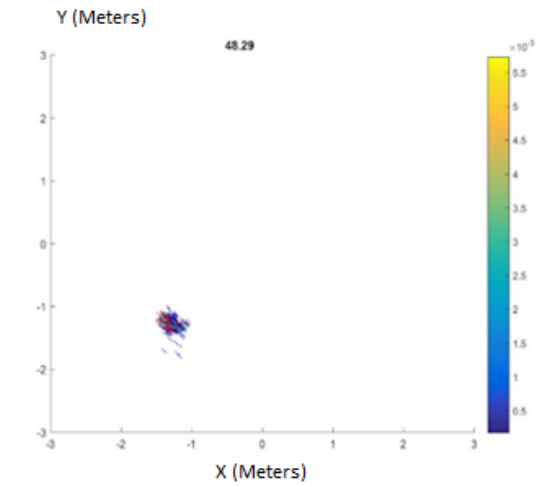
roslaunch turtlebot_gazebo turtlebot_world.launch
world_file:=/opt/ros/indigo/share/turtlebot_gazebo/worlds/playground.world
start_driving_robot.sh
#!/bin/bash
clear
rosbag play /home/ebrenna8/Turtlebot_World_Driving.bag
rosnode kill -a
killall -9 gzserver
start_amcl.sh
source /home/ebrenna8/amcl_overlay/devel/setup.bash;
roslaunch turtlebot_gazebo amcl_demo.launch
map_file:=/opt/ros/indigo/share/turtlebot_gazebo/maps/playground.yaml initial_pose_x:=0
initial_pose_y:=0 initial_pose_a:=0

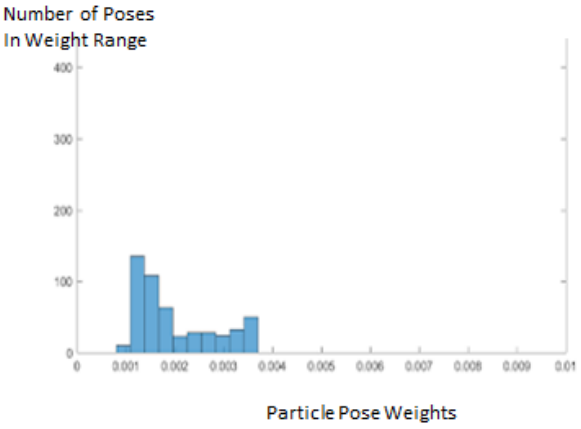
Appendix E: Data Collected from Robot Kidnapping Simulation Environment

Kidnap Type	Train/Test Set	Kidnap Time (sec. after start of trial)	Number of Trials Collected	Average Number of Time Steps in Trials	Number of Time Steps Used in Training/Testing
None-Experiment Control	Control	n/a	10	180	n/a
Major: On-Map	Train	30	3	11	33
		50	1	30	30
		75	1	45	45
Major: On-Map	Test	30	4	11	44
		40	2	24	48
		50	2	30	60
		75	4	45	180
Minor: Theta	Train	30	11	9	99
Minor: Theta	Test	30	5	9	45
		45	1	25	25
		75	3	54	162
Minor: (X,Y)	Train	30	2	11	22
		45	2	22	44
		75	2	53	106
Minor: (X, Y)	Test	30	1	12	12
		45	2	32	64
		75	1	60	60
Prolonged Disturbance	Train	30	4	10	40
		40	3	10	30
Prolonged Disturbance	Test	30	1	5	5
TOTAL			**65**		**1154**

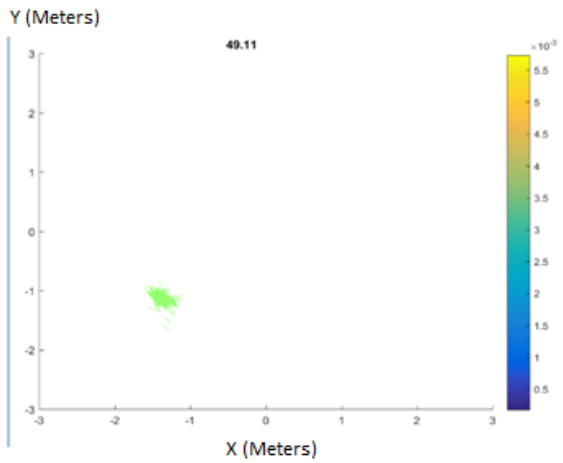
Appendix F

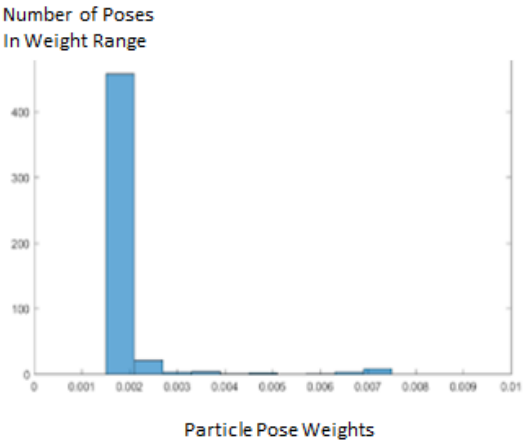
Prior to a Kidnapping Event





After a Kidnapping Event





Appendix G

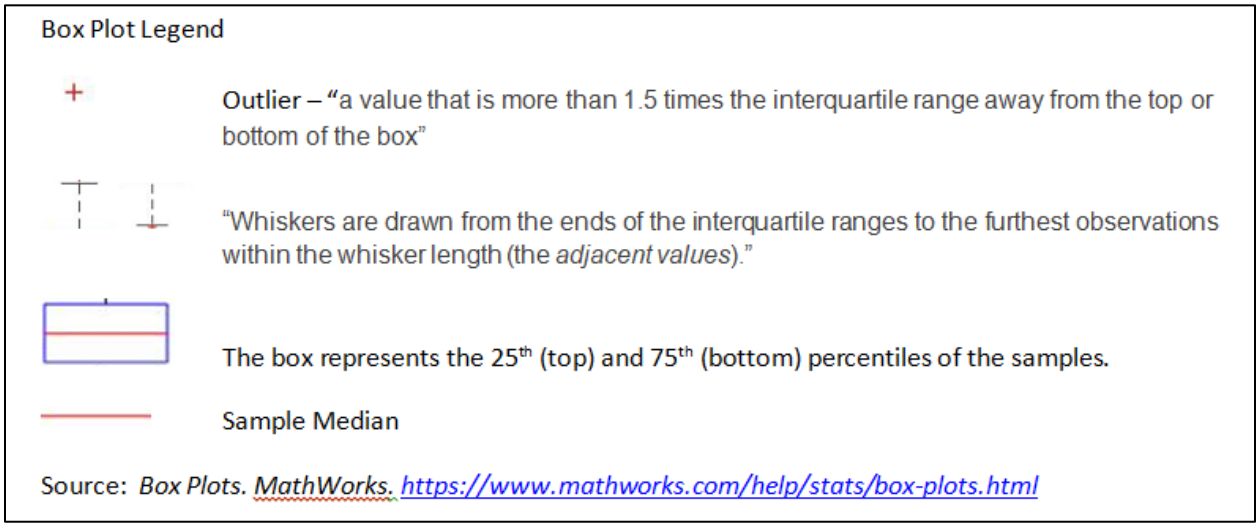


Figure G1: Elements on a Box and Whisker Plot

Appendix H

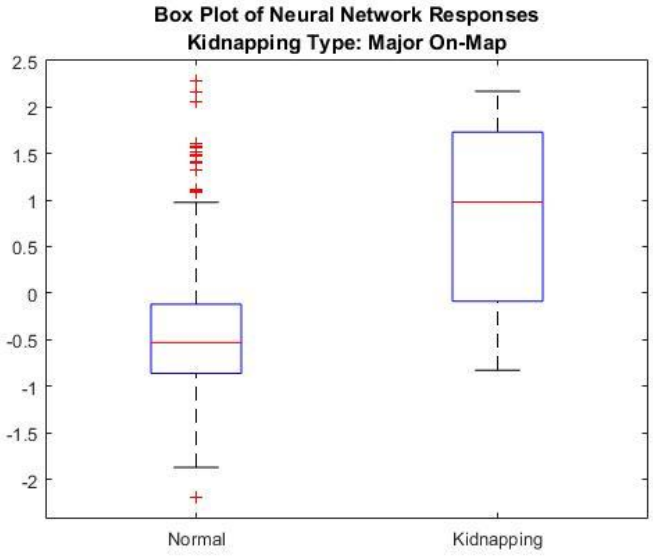


Figure H1: Box and Whisker plot comparing the statistics of the Neural Network Responses for Normal and Major Kidnapping Scenarios

Table H1: Statistical Information about Major On-Map Kidnapping Neural Network Trials

Category	Mean	Median	25^{th} Percentile	75^{th} Percentile
Major Kidnapping	0.8675	0.9771	-0.867	1.7283
Normal	-0.3655	-0.5298	-0.8610	-0.1203

Table H2: Evaluation of Threshold Candidates for Major (On-Map) Kidnapping datasets

Threshold Derived From:	**Threshold Value:**	**False Positives/Percent False Positives (based on total timesteps surveyed)**	**False Negatives / Percent False Negatives**	**Average Time to Detection (Seconds)**	**Average Time Steps to Detection**	**Total Time Steps from Trial Start to Kidnap Detection**
Minimum Post-Kidnap Neural Network Response	1.2	17 / 6.55%	14 / 4.78%	1.12	0.11	331
Median	0.9771	16 / 4.79%	16 / 4.79%	2.5	1.6	334
25th Percentile of Statistical Distribution of Kidnap NN responses	-0.867	94 / 29.47%	0 / 0%	1.744	0	319

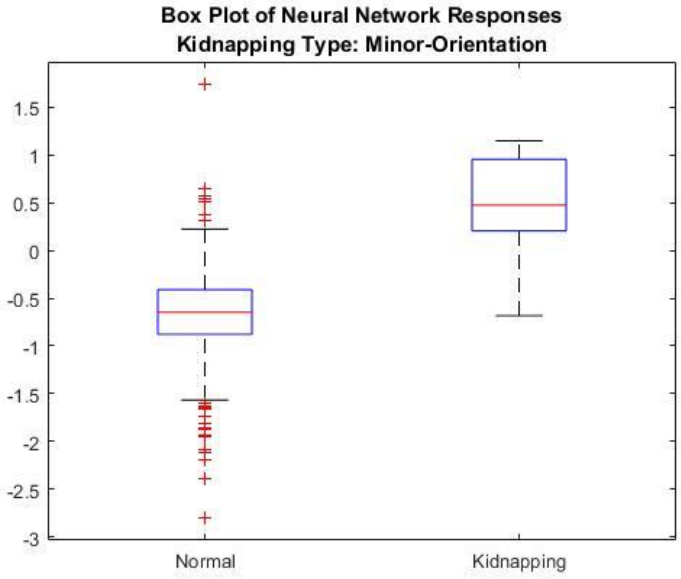


Figure H2: Box and Whisker plot comparing the Neural Network Responses for Normal and Minor-Orientation Kidnapping Scenarios

Table H3: Statistical Information about Normal and Minor - Orientation Kidnapping scenarios

Category	Mean	Median	25^{th} Percentile	75^{th} Percentile
Minor Kidnapping (Orientation)	0.4496	0.4747	0.2048	0.9547
Normal	-0.6628	-0.6476	-0.8787	-0.4110

Table H4: Evaluation of Threshold Candidates for Minor (Orientation) Kidnapping datasets

Threshold Derived From:	**Threshold Value:**	**False Positives/Percent False Positives (based on total time steps surveyed)**	**False Negatives / Percent False Negatives**	**Average Time to Detection (Seconds)**	**Average Time Steps to Detection**	**Total Time Steps from Trial Start to Kidnap Detection**
Minimum Post-Kidnap Neural Network Response	0.15	16 / 4.93%	1 / 1.01%	1.12	0.11	232
Median	0.4747	9 / 3.78%	12 / 5.04%	1.47	1.33	238
25th Percentile of Statistical Distribution of Kidnap NN responses	0.2048	110 / 48.46%	0 / 0.00%	1.47	0	227

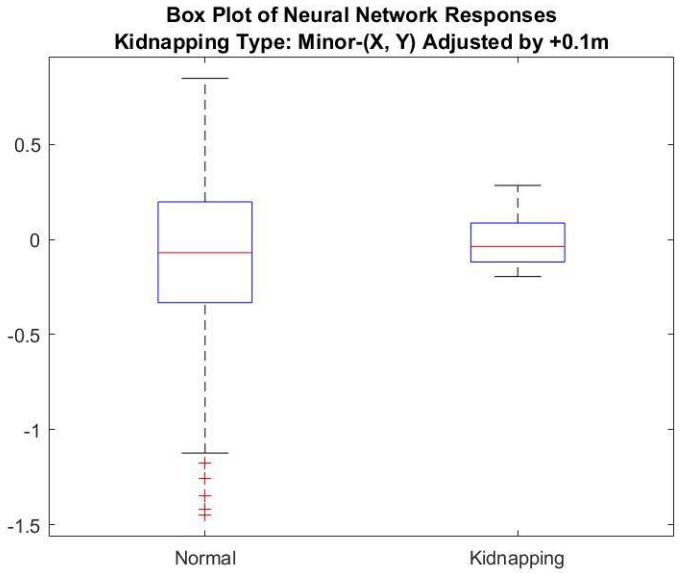


Figure H3: Box and Whisker plot comparing the Neural Network Responses for Normal and Kidnapping Scenarios

Table H5: Statistical Information about Normal and Minor - (X,Y) Kidnapping scenarios

Category	Mean	Median	25th Percentile	75th Percentile
Minor Kidnapping (*X*, *Y*)	-0.0059	-0.0368	-0.1183	0.0865
Normal	-0.1255	-0.0694	-0.3318	0.1979

Table H6: Evaluation of Threshold Candidates for Minor (X, Y) Kidnapping datasets

Threshold Derived From:	**Threshold Value:**	**False Positives/Percent False Positives (based on total time steps surveyed)**	**False Negatives / Percent False Negatives**	**Average Time to Detection (Seconds)**	**Average Time Steps to Detection**	**Total Time Steps from Trial Start to Kidnap Detection**
Minimum Post-Kidnap Neural Network Response	-0.4797	87 / 82.08%	0 / 0.00%	1.56	0	106
Median	-0.0368	54 / 50.94%	1 / 0.94%	1.56	0.2	106
25th Percentile of Statistical Distribution of Kidnap NN responses	-0.1183	81 / 76.42%	0 / 0.00%	1.56	0	106

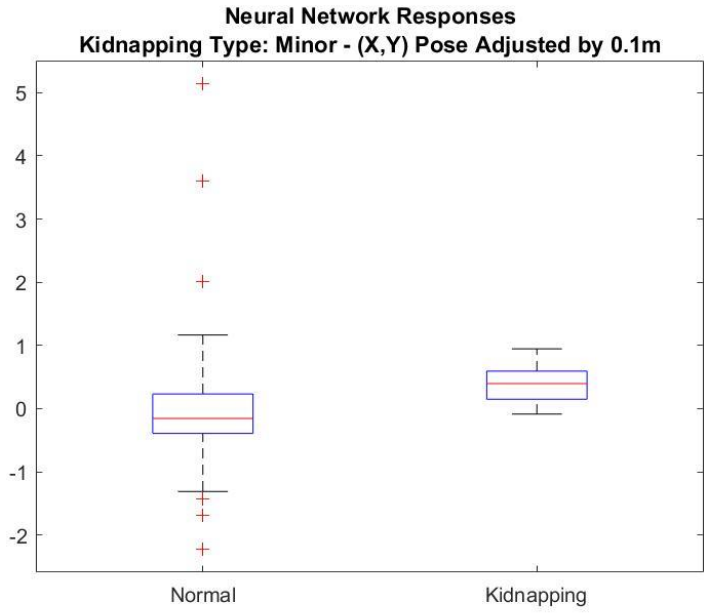


Figure H4: Box and Whisker plot comparing the Neural Network Responses for Normal and Kidnapping Scenarios

Table H7: Statistical Information about Normal and Minor - (*X,Y*) Kidnapping scenarios

Category	Mean	Median	25th Percentile	75th Percentile
Minor Kidnapping (*X*, *Y*) Using Major Kidnapping Model	0.3943	0.3992	0.1513	0.5948
Normal	-0.0313	-0.1538	-0.3908	0.2339

Table H8: Evaluation of Threshold Candidates for Minor (X, Y) Kidnapping datasets when evaluated against the Major Kidnapping model

Threshold Derived From:	**Threshold Value:**	**False Positives/Percent False Positives (based on total time steps surveyed)**	**False Negatives / Percent False Negatives**	**Average Time to Detection (Seconds)**	**Average Time Steps to Detection**	**Total Time Steps from Trial Start to Kidnap Detection**
Minimum Post-Kidnap Neural Network Response	0.4	0 / 0.00%	17.42%	2.74	1.75	106
Median	0.3992	21 / 19.81%	2 / 1.89%	1.56	0.4	106
25th Percentile of Statistical Distribution of Kidnap NN responses	0.1513	30 / 28.30%	0 / 0.00%	1.56	0.2	106

Appendix I

It should be noted that the Prolonged Disturbance kidnapping method would be a more accurate simulation if the physical properties of the robot were adjusted, such as reducing the radius of the robot's tires mid-trail to simulate a flat tire or changing the friction property of the floor to simulate a sticky floor. These changes to the Gazebo elements were researched, but attempts to adjust the properties of a model or environment during the simulation failed. Known Gazebo bugs related to deleting and respawning a robot model during a Gazebo simulation were encountered when trying to swap out robot models with various property changes such as a reduction in wheel radius to represent a flat tire. Initial research indicated that it would be possible to set the properties of Gazebo model elements, such as wheel radius, during a Gazebo simulation. Experiments to test this with the Turtlebot Gazebo model found that while some Gazebo models' properties can be modified during a Gazebo simulation, the Turtlebot robot model properties cannot be modified. Therefore, the geometric approach was developed for approximating the effects of physical changes on the robot.

Appendix J

This work derived a mathematical model of simulating scenarios of a reduction in wheel velocity on one of the differential drive robot's wheels while keeping the other wheel at the intended velocity. This scenario challenges the accuracy of the robot localization system. Examples of such scenarios include driving the robot on a flat tire or driving on a course in which the robot encounters Wheel Slip conditions. To change the robot's Gazebo position as it moved, a Gazebo plugin function paused the Gazebo simulation at specified time intervals. While the simulation was paused, the Gazebo robot model's pose was reset to a calculated location. The location to which the robot was repositioned was calculated by considering the current Gazebo pose and the previous Gazebo pose to provide context for the robot model's recent motion history. The derivation of the method is as follows.

Page 118 of Probabilistic Robotics by Thrun, Burgard, and Fox[5] notes that in the book's kinematic equations, the robot's configuration "is described by six variables, its three-dimensional Cartesian coordinates and its three Euler angles (roll, pitch, yaw), relative to an external coordinate frame." In ROS, tf messages deliver updates about the robot's coordinates in the form of tf::TransformStamped messages, which contain a vector of the x, y, and z Cartesian coordinates in the "odom" world frame, and a Quaternion message type containing the robot's orientation information.

The kinematic equations of Probabilistic Robotics use θ in units of radians for the robot's bearing, according to page 119. Therefore, to use the Quaternion data from the tf message in the kinematic equations, the quaternion must be translated to Roll, Pitch, Yaw form in units of radians. The tf::Matrix3x3 class and its associated method, getRPY, are used to convert the tf::StampedTransform quaternion to Roll, Pitch, Yaw in units of radians and obtain θ from the Yaw parameter in units of radians to be used in the kinematic equations. On page 127 of

Probabilistic Robotics, Figure 5.9 shows the following equation for finding the pose update in a time step based on a constant velocity during the time step:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{w}sin\theta + \frac{v}{w}\sin(\theta + w\Delta t) \\ \frac{v}{w}cos\theta - \frac{v}{w}\cos(\theta + w\Delta t) \\ w\Delta t \end{pmatrix}$$

In this study, Δt is considered the publication interval of tf messages pertaining to the right and left wheel links, and since the Gazebo callback function calculates position updates each time this information is received from the tf message publisher, Δt is set to 1 second.

The robot's current pose $\begin{pmatrix} \mathrm{x} \\ \mathrm{y} \\ \theta \end{pmatrix}$ is obtained from the tf messages' transform between the odom frame and the base_footprint frame. Variables v and w are the robot's translational and rotational (angular) velocities in the world frame (i.e.,the odom frame) in units of meters/second and radians, respectively, per ROS standards. Likewise, in the kinematic equations, θ is the robot's orientation in the world frame, in units of radians, as inferred from page 119 of Probabilistic Robotics.

Rotational velocity is obtained by determining the robot's change in bearing, $\Delta\,\theta$, between the previous timestep, x_{t-1} and the current timestep, x_t. Per page 123 of Probabilistic Robotics,

$$\mu = \frac{1}{2}\left(\frac{(x - x')cos\theta + (y - y')sin\theta}{(y - y')cos\theta - (x - x')sin\theta}\right)$$

$$x^* = \frac{x + x'}{2} + \mu(y - y')$$

$$y^* = \frac{y + y'}{2} + \mu(x' - x)$$

$$\Delta\,\theta = atan2(y' - y^*, x' - x^*) - atan2(y - y^*, x - x^*)$$

where x' and y' represent the robot's pose at x_t, and x and y represent the robot's pose at x_{t-1}.

Given $\Delta\,\theta$, the equation for *w*, the rotational or angular velocity of the robot, is

$$w = \frac{\Delta\,\theta}{\Delta t}$$

Knowing angular velocity enables the calculation of translational velocity, v, as expressed on page 123 of Probabilistic Robotics:

$$r^* = \sqrt{(x - x^*)^2 + (y - y^*)^2}$$
$$v = w * r^*$$

For the purposes of this study, these equations must be rewritten so that the equations for the robot's translational and rotational velocities are broken into their component parts, i.e., the left and the right wheels, individually. Page 52 of Introduction to Autonomous Mobile Robots by Seigwart and Nourbakhsh demonstrates that the differential drive robot's rotational velocity is equal to the sum of the rotational velocity calculated for the left wheel and the right wheel's rotational velocity. This principle is used to rewrite the kinematic equations of Probabilistic Robotics for a differential drive robot as such:

$$\mu_{left} = \frac{1}{2}\left(\frac{(x_{left} - x_{left}{}')cos\theta + (y_{left} - y_{left}{}')sin\theta}{(y_{left} - y_{left}{}')cos\theta - (x_{left} - x_{left}{}')sin\theta}\right)$$
$$x_{left}{}^* = \frac{x_{left} + x_{left}{}'}{2} + \mu_{left}(y_{left} - y_{left}{}')$$
$$y_{left}{}^* = \frac{y_{left} + y_{left}{}'}{2} + \mu_{left}(x_{left}{}' - x_{left})$$
$$\Delta\,\theta_{left} = atan2(y_{left}{}' - y_{left}{}^*, x_{left}{}' - x_{left}{}^*) - atan2(y_{left} - y_{left}{}^*, x_{left} - x_{left}{}^*)$$
$$w_{left} = \frac{\Delta\,\theta_{left}}{\Delta t}$$
$$r^*{}_{left} = \sqrt{(x_{left} - x_{left}{}^*)^2 + (y_{left} - y_{left})^2}$$
$$v_{left} = w_{left} * r^*{}_{left}$$

The equations to find w_{right} and v_{right} for the right wheel are written similarly, using $x_{right} + x_{right}{}'$. The motion estimate is made using the translational and rotational velocities for the two wheels, adapting the approach of Probabilistic Robotics page 127 to use the left and right velocities:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_{right} + v_{left}}{w_{right} + w_{left}} sin\theta + \frac{v_{right} + v_{left}}{w_{right} + w_{left}} \sin(\theta + \Delta t(w_{right} + w_{left})) \\ \frac{v_{right} + v_{left}}{w_{right} + w_{left}} cos\theta - \frac{v_{right} + v_{left}}{w_{right} + w_{left}} \cos(\theta + \Delta t(w_{right} + w_{left})) \\ (w_{right} + w_{left})\Delta t \end{pmatrix}$$

Breaking the velocity component of the motion update into components representing the left and right wheel velocities of a differential robot can be used to create equations that simulate the robot's pose that would have resulted had the robot's left or right wheel been travelling at a different velocity. In a Flat Tire scenario, one of the robot's wheels operates at a reduced velocity than the other wheel. This reduction in velocity affects the robot's pose, because "Differential drive vehicles are very sensitive to slight changes in velocity in each of the wheels. Small errors in the relative velocities between the wheels can affect the robot trajectory."[45] The effects on the robot's motion caused by a reduction in velocity of one of its wheels can be approximated by reducing the radius of the circle on which the robot turns as the robot moves.

The two starting pieces of information for the calculation are (x_t, y_t, Θ_t), representing the robot's current Gazebo pose, and (x_{t-1}, y_{t-1}, Θ_{t-1}), representing the robot's Gazebo pose in the previous Simulation Time time step. The Euclidean distance *r* between (x_t, y_t, Θ_t) and (x_{t-1}, y_{t-1}, Θ_{t-1}) is calculated.

$$r = \sqrt{((x_t - x_{t-1})^2 + (y_t - y_{t-1})^2)}$$

Next, equations for the circles centered at the current and previous Gazebo poses passing through the other Gazebo pose are found. Equations for the circles *C1* and *C2* are in Center-Radius form as:

$$C1 = r^2 = (x - x_{t-1})^2 + (y - y_{t-1})^2$$
$$C2 = r^2 = (x - x_t)^2 + (y - y_t)^2$$

The equations *C1* and *C2* are solved for the (*x*, *y*) coordinates of the two intersection points of circles *C1* and *C2*, represented as p_1 and p_2 (Figure J1).

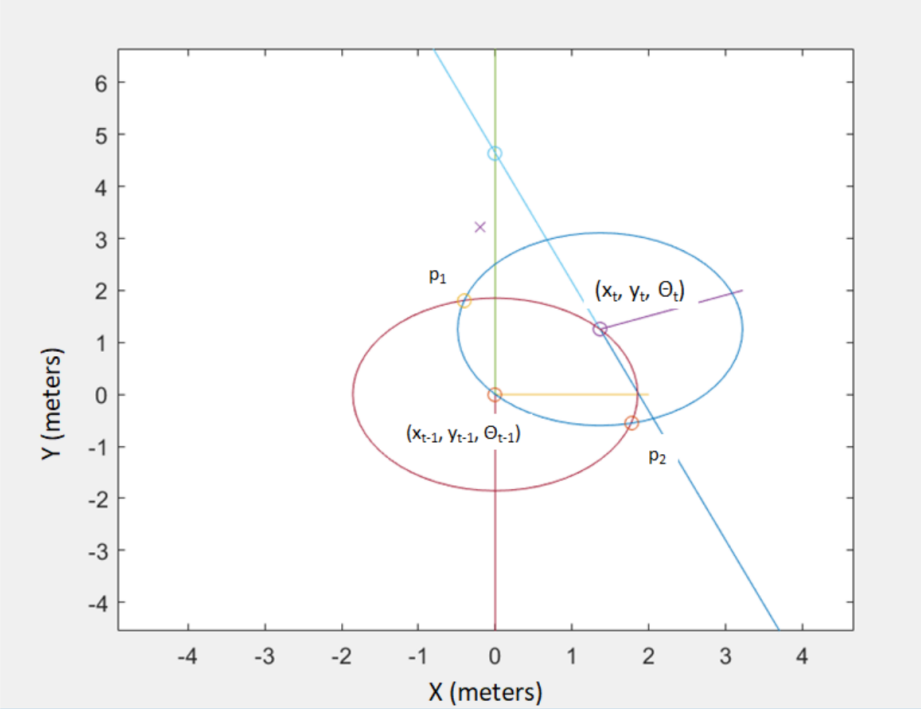


Figure J1: Intercepts of the circles C1 and C2

A circular arc a_1 exists on circle *C1* between p_1 and (x_t, y_t). Another circular arc, a_2, exists on C1 between (x_t, y_t) and p_2 (Figure J1). One of these arcs corresponds to the robot's motion from time step *t-1* to *t*. To determine whether the robot's motion is represented by a_1 or a_2, the intersection of the tangent lines of (x_t, y_t, Θ_t) and $(x_{t-1}, y_{t-1}, \Theta_{t-1})$ is calculated (Figure J2). The arc *A* with an end point nearest to the intersection of the tangent lines is chosen as the arc that represents the robot's motion (Figure J3). The arc length L_A of arc *A* is calculated using (x_{t-1}, y_{t-1}) and the arc endpoints.

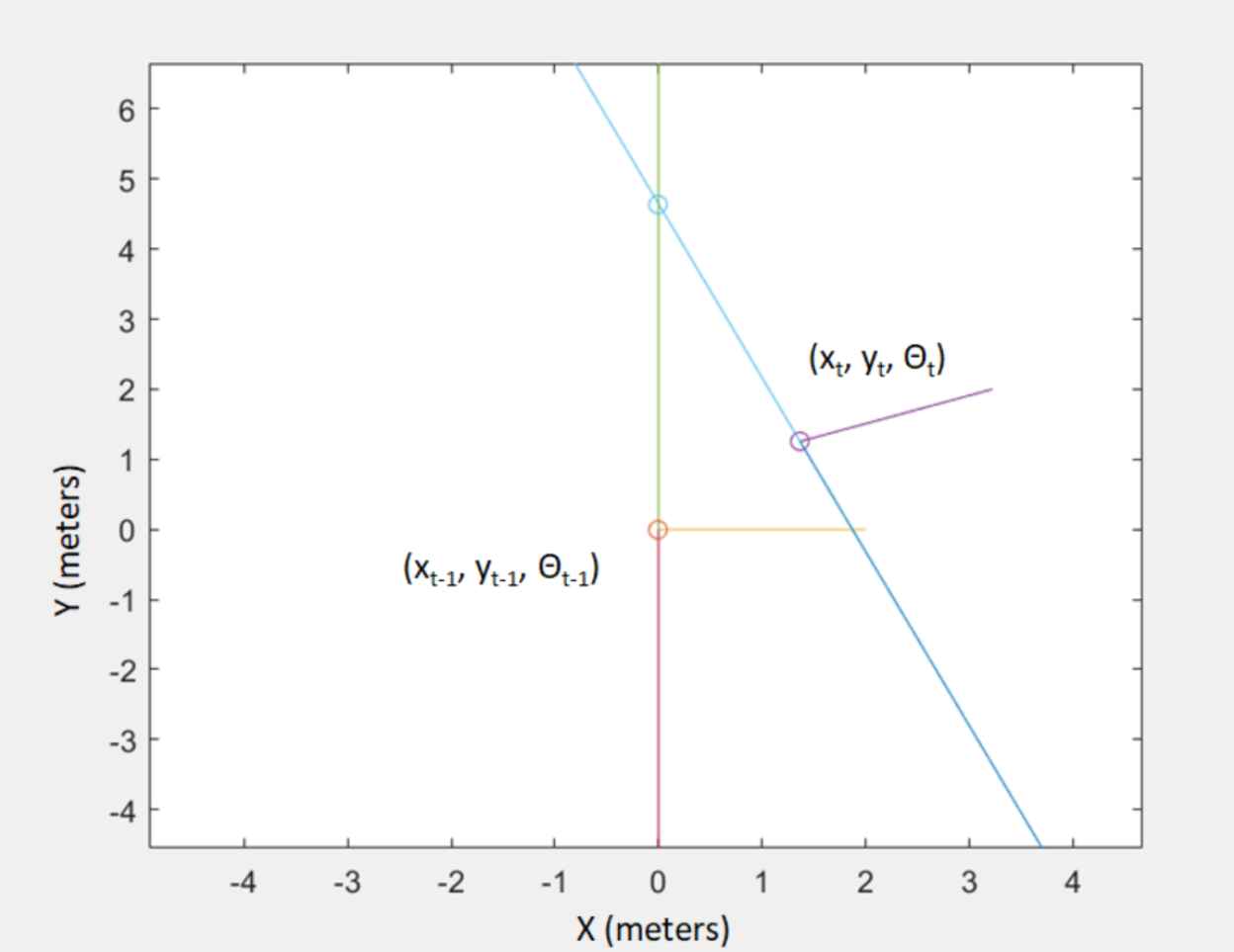


Figure J2: Tangent lines are calculated for the robot poses

A new pose that represents a change in the robot's motion that is some percent value *z* of the robot's motion from time *t-1* to *t* can be calculated by multiplying the radius of *C1* by *z*. The resulting radius value is r_1. Using r_1 and L_A, an arc angle A_A is calculated for the arc with the same endpoints as arc A, but with radius r_1.

$$r_1 = z * r$$

$$A_A = \frac{L_A * 2 * \pi}{r_1}$$

The center of the center of the circle (x_1, y_1) associated with A_A and $r_1$is calculated by finding the two-argument arctangent (atan2) for (x_{t-1}, y_{t-1}) and (x_t, y_t).

$$tmpang = \text{atan}\big((y_{t-1} - y_t), (x_{t-1} - x_t)\big)$$

$$x_1 = x_{t-1} + (r_1 - r) * \cos(tmpang)$$

$$y_1 = y_{t-1} + (r_1 - r) * \sin(tmpang)$$

The robot's new pose can be calculated based on the arc angle A_A, the center of the new circle (x_1, y_1), and the new circle's radius r_1.

$$\begin{aligned} newX &= x_1 + r_1 * cos(ang1 + A_A); \\ newY &= y_1 + (r_1 * sin(ang1 + A_A) \\ newTheta &= \Theta_{t-1} * (2 * pi/360) + A_A \end{aligned}$$

where ang1 is

$$\begin{aligned} projX1 &= x_{t-1} - centX; \\ projY1 &= y_{t-1} - centY; \\ ang1 &= atan2(projY1, projX1); \end{aligned}$$

and (*centX, centY*) is a point located at the midpoint of the line segment drawn between the intersection of the tangent lines of (x_{t-1}, y_{t-1}) and (x_t, y_t) and the arc endpoint closest to the intersection of the tangent lines (Figure J3). See the code in Appendix L for the full implementation.

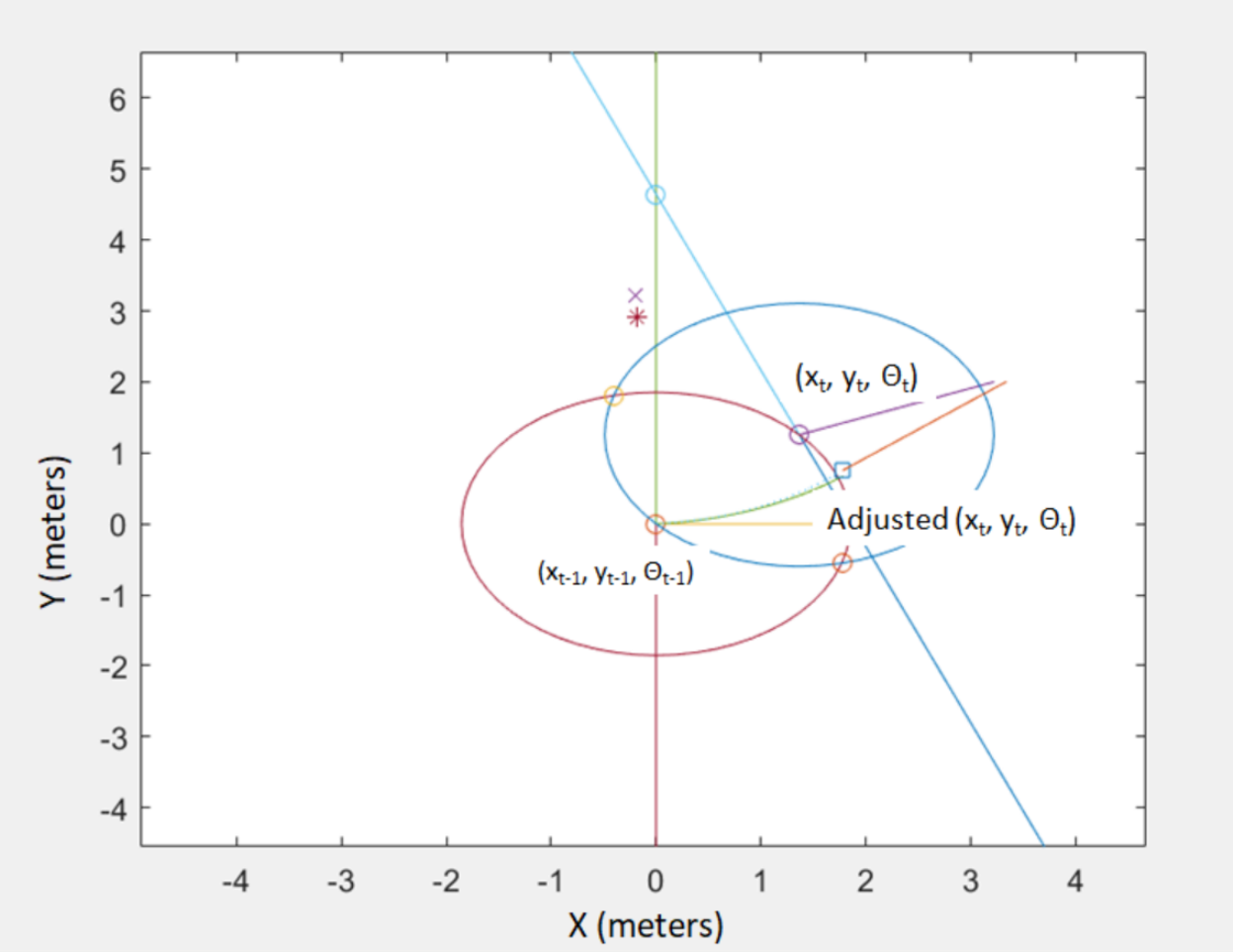


Figure J3: An adjusted pose is calculated to simulated the motion achieved if one of the wheel velocity had been different

As an example, assume the robot's pose at *t-1* is (0, 0, 0) and motion commands are issued at time *t-1* that change the robot's pose to (1, 1, 0.34) at time *t*. The equations above can be used to adapt the robot's pose to a location that would have resulted if the robot's left wheel was operating at a velocity that was 90% of the right wheel's velocity. Providing (0, 0, 0) for $(x_{t-1}, y_{t-1}, \Theta_{t-1})$, (1.37, 1.25, 21.9) for (x_t, y_t, Θ_t), and 90% or 0.9 for *z* to the equations yields (1.77, 0.75, 0.67) as the robot's adjusted pose (Figure J4).

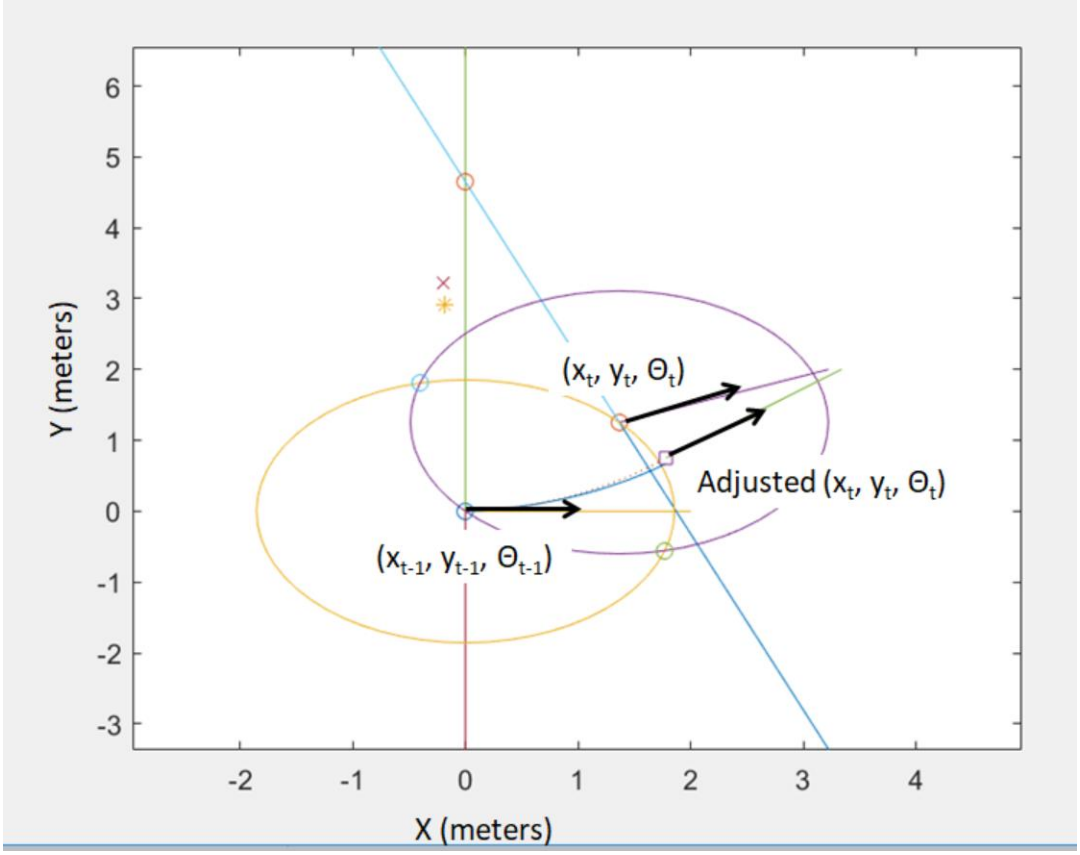


Figure 4: Example of the pose adjustment created by this method

Appendix K

Table 1: Performance Metrics for Proposed Kidnap Detection/Classification Method Compared with Other Methods

Speed of Detection					
Team	**Metric**	**Value**	**Proposed Method's Average Value - Hallway**	**Proposed Method's Average Value - Gazebo**	
Lee et al. [1]	Speed of Detection	68.8 seconds	2.74 s	2.7 s	
Tian et al. [10]	Speed to Detection	13 µs	2.74 s	2.7 s	
Ward et al. [4]	Wheel Slip Speed of Detection	0.4 seconds	2.74 s	2.7 s	
Detection Accuracy					
Yi et al. [16]	Major Kidnapping Detection Accuracy	35%	72% Kidnapping Event Detection, 20% Correct Classification	88.67% Kidnapping Event Detection	
Zhang et al. [13]]	Major Kidnapping Detection Accuracy	40%	72% Kidnapping Event Detection, 20% Correct Classification	88.67% Kidnapping Event Detection	
Bukhori et al. [12]	Major Kidnapping Detection Accuracy	70%	72% Event Detection, 20% Correct Classification	88.67% Kidnapping Event Detection	
False Negatives					
Team	**Metric**	**Value**	**Proposed Method's Value**	**Proposed Method's Average Value - Gazebo**	
Campbell et al. [11]	Wheel Slip False Negatives	18%	24.64%	1.86%	
Campbell et al. [11]	Major Kidnapping False Negatives	0.30%	28%	2%	

Appendix L

Modifications to the ROS amcl package's amcl_node.cpp:

```
/*
 *  Copyright (c) 2008, Willow Garage, Inc.
 *  All rights reserved.
 *
 *  This library is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU Lesser General Public
 *  License as published by the Free Software Foundation; either
 *  version 2.1 of the License, or (at your option) any later version.
 *
 *  This library is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 *  Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this library; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 */

/* Author: Brian Gerkey */

#include <algorithm>
#include <vector>
#include <map>
#include <cmath>

#include <boost/bind.hpp>
#include <boost/thread/mutex.hpp>

// Signal handling
#include <signal.h>

#include "map/map.h"
#include "pf/pf.h"
#include "sensors/amcl_odom.h"
#include "sensors/amcl_laser.h"

#include "ros/assert.h"

// roscpp
```

```
#include "ros/ros.h"

// Messages that I need
#include "sensor_msgs/LaserScan.h"
#include "geometry_msgs/PoseWithCovarianceStamped.h"
#include "geometry_msgs/PoseArray.h"
#include "geometry_msgs/Pose.h"
#include "nav_msgs/GetMap.h"
#include "nav_msgs/SetMap.h"
#include "std_srvs/Empty.h"


//Custom messages that I need
#include "/home/ebrenna8/amcl_overlay/devel/include/amcl/testmsg.h"
#include "/home/ebrenna8/amcl_overlay/devel/include/amcl/numsamples_msg.h"
#include "/home/ebrenna8/amcl_overlay/devel/include/amcl/max_weight_hyp_msg.h"
#include "/home/ebrenna8/amcl_overlay/devel/include/amcl/cluster_count_msg.h"
#include "/home/ebrenna8/amcl_overlay/devel/include/amcl/delta_pose_msg.h"
#include "/home/ebrenna8/amcl_overlay/devel/include/amcl/filter_covariance_msg.h"
//#include "/home/ebrenna8/amcl_overlay/devel/include/amcl/cloud_weight_msg.h"


// For transform support
#include "tf/transform_broadcaster.h"
#include "tf/transform_listener.h"
#include "tf/message_filter.h"
#include "tf/tf.h"
#include "message_filters/subscriber.h"

// Dynamic_reconfigure
#include "dynamic_reconfigure/server.h"
#include "amcl/AMCLConfig.h"

// Allows AMCL to run from bag file
#include <rosbag/bag.h>
#include <rosbag/view.h>
#include <boost/foreach.hpp>

#define NEW_UNIFORM_SAMPLING 1

using namespace amcl;

// Pose hypothesis
typedef struct
{
  // Total weight (weights sum to 1)
```

```
  double weight;

  // Mean of pose esimate
  pf_vector_t pf_pose_mean;

  // Covariance of pose estimate
  pf_matrix_t pf_pose_cov;

} amcl_hyp_t;

static double
normalize(double z)
{
  return atan2(sin(z),cos(z));
}
static double
angle_diff(double a, double b)
{
  double d1, d2;
  a = normalize(a);
  b = normalize(b);
  d1 = a-b;
  d2 = 2*M_PI - fabs(d1);
  if(d1 > 0)
    d2 *= -1.0;
  if(fabs(d1) < fabs(d2))
    return(d1);
  else
    return(d2);
}

static const std::string scan_topic_ = "scan";

class AmclNode
{
  public:
    AmclNode();
    ~AmclNode();

    /**
     * @brief Uses TF and LaserScan messages from bag file to drive AMCL instead
     */
    void runFromBag(const std::string &in_bag_fn);

    int process();
    void savePoseToServer();
```

```
  private:
    tf::TransformBroadcaster* tfb_;

    // Use a child class to get access to tf2::Buffer class inside of tf_
    struct TransformListenerWrapper : public tf::TransformListener
    {
      inline tf2_ros::Buffer &getBuffer() {return tf2_buffer_;}
    };

    TransformListenerWrapper* tf_;

    bool sent_first_transform_;

    tf::Transform latest_tf_;
    bool latest_tf_valid_;

    // Pose-generating function used to uniformly distribute particles over
    // the map
    static pf_vector_t uniformPoseGenerator(void* arg);
#if NEW_UNIFORM_SAMPLING
    static std::vector<std::pair<int,int> > free_space_indices;
#endif
    // Callbacks
    bool globalLocalizationCallback(std_srvs::Empty::Request& req,
                                    std_srvs::Empty::Response& res);
    bool nomotionUpdateCallback(std_srvs::Empty::Request& req,
                                    std_srvs::Empty::Response& res);
    bool setMapCallback(nav_msgs::SetMap::Request& req,
                        nav_msgs::SetMap::Response& res);

    void laserReceived(const sensor_msgs::LaserScanConstPtr& laser_scan);
    void initialPoseReceived(const geometry_msgs::PoseWithCovarianceStampedConstPtr&
msg);
    void handleInitialPoseMessage(const geometry_msgs::PoseWithCovarianceStamped& msg);
    void mapReceived(const nav_msgs::OccupancyGridConstPtr& msg);

    void handleMapMessage(const nav_msgs::OccupancyGrid& msg);
    void freeMapDependentMemory();
    map_t* convertMap( const nav_msgs::OccupancyGrid& map_msg );
    void updatePoseFromServer();
    void applyInitialPose();

    double getYaw(tf::Pose& t);

    //parameter for what odom to use
```

```
    std::string odom_frame_id_;

    //paramater to store latest odom pose
    tf::Stamped<tf::Pose> latest_odom_pose_;

    //parameter for what base to use
    std::string base_frame_id_;
    std::string global_frame_id_;

    bool use_map_topic_;
    bool first_map_only_;

    ros::Duration gui_publish_period;
    ros::Time save_pose_last_time;
    ros::Duration save_pose_period;

    geometry_msgs::PoseWithCovarianceStamped last_published_pose;

    map_t* map_;
    char* mapdata;
    int sx, sy;
    double resolution;

    message_filters::Subscriber<sensor_msgs::LaserScan>* laser_scan_sub_;
    tf::MessageFilter<sensor_msgs::LaserScan>* laser_scan_filter_;
    ros::Subscriber initial_pose_sub_;
    std::vector< AMCLLaser* > lasers_;
    std::vector< bool > lasers_update_;
    std::map< std::string, int > frame_to_laser_;

    // Particle filter
    pf_t *pf_;
    double pf_err_, pf_z_;
    bool pf_init_;
    pf_vector_t pf_odom_pose_;
    double d_thresh_, a_thresh_;
    int resample_interval_;
    int resample_count_;
    double laser_min_range_;
    double laser_max_range_;

    //Nomotion update control
    bool m_force_update;  // used to temporarily let amcl update samples even when no motion
occurs...

    AMCLOdom* odom_;
```

```
  AMCLLaser* laser_;

  ros::Duration cloud_pub_interval;
  ros::Time last_cloud_pub_time;

  // For slowing play-back when reading directly from a bag file
  ros::WallDuration bag_scan_period_;

  void requestMap();

  // Helper to get odometric pose from transform system
  bool getOdomPose(tf::Stamped<tf::Pose>& pose,
                   double& x, double& y, double& yaw,
                   const ros::Time& t, const std::string& f);

  //time for tolerance on the published transform,
  //basically defines how long a map->odom transform is good for
  ros::Duration transform_tolerance_;

  ros::NodeHandle nh_;
  ros::NodeHandle private_nh_;
  ros::Publisher pose_pub_;
  ros::Publisher particlecloud_pub_;
  ros::Publisher particlecloud_pub_PreResample;
  ros::Publisher test_pub_;
  ros::Publisher numsamples_pub_;
  ros::Publisher max_weight_hyp_pub_;
  ros::Publisher cluster_count_pub_;
  ros::Publisher delta_pose_pub_;
  ros::Publisher filter_covariance_pub_;
  ros::Publisher cloud_weight_pub_;
  ros::Publisher cloud_weight_pub_beforeUpdateSensor;

  ros::ServiceServer global_loc_srv_;
  ros::ServiceServer nomotion_update_srv_; //to let amcl update samples without requiring
motion
  ros::ServiceServer set_map_srv_;
  ros::Subscriber initial_pose_sub_old_;
  ros::Subscriber map_sub_;

  amcl_hyp_t* initial_pose_hyp_;
  bool first_map_received_;
  bool first_reconfigure_call_;

  boost::recursive_mutex configuration_mutex_;
  dynamic_reconfigure::Server<amcl::AMCLConfig> *dsrv_;
```

```
    amcl::AMCLConfig default_config_;
    ros::Timer check_laser_timer_;

    int max_beams_, min_particles_, max_particles_;
    double alpha1_, alpha2_, alpha3_, alpha4_, alpha5_;
    double alpha_slow_, alpha_fast_;
    double z_hit_, z_short_, z_max_, z_rand_, sigma_hit_, lambda_short_;
  //beam skip related params
    bool do_beamskip_;
    double beam_skip_distance_, beam_skip_threshold_, beam_skip_error_threshold_;
    double laser_likelihood_max_dist_;
    odom_model_t odom_model_type_;
    double init_pose_[3];
    double init_cov_[3];
    laser_model_t laser_model_type_;
    bool tf_broadcast_;

    void reconfigureCB(amcl::AMCLConfig &config, uint32_t level);

    ros::Time last_laser_received_ts_;
    ros::Duration laser_check_interval_;
    void checkLaserReceived(const ros::TimerEvent& event);
};

std::vector<std::pair<int,int> > AmclNode::free_space_indices;

#define USAGE "USAGE: amcl"

boost::shared_ptr<AmclNode> amcl_node_ptr;

void sigintHandler(int sig)
{
  // Save latest pose as we're shutting down.
  amcl_node_ptr->savePoseToServer();
  ros::shutdown();
}

int
main(int argc, char** argv)
{
  ros::init(argc, argv, "amcl");
  ros::NodeHandle nh;
          ROS_INFO("EEEEEEEEEEEEEELLLLLLLLLLLLLLLLLLLIIIIIIIIIIIIIIIIIIZZZZZZZ
ZZZZZZZZZAAAAAAAAAAAABBBBBBBBBBBBBBBBEEEEEEEEEEEEEETHHHHHHHH
HHHH");
  // Override default sigint handler
```

```
  signal(SIGINT, sigintHandler);

  // Make our node available to sigintHandler
  amcl_node_ptr.reset(new AmclNode());

  if (argc == 1)
  {
    // run using ROS input
    ros::spin();
  }
  else if ((argc == 3) && (std::string(argv[1]) == "--run-from-bag"))
  {
    amcl_node_ptr->runFromBag(argv[2]);
  }

  // Without this, our boost locks are not shut down nicely
  amcl_node_ptr.reset();

  // To quote Morgan, Hooray!
  return(0);
}

AmclNode::AmclNode() :
        sent_first_transform_(false),
        latest_tf_valid_(false),
        map_(NULL),
        pf_(NULL),
        resample_count_(0),
        odom_(NULL),
        laser_(NULL),
                private_nh_("~"),
        initial_pose_hyp_(NULL),
        first_map_received_(false),
        first_reconfigure_call_(true)
{
  boost::recursive_mutex::scoped_lock l(configuration_mutex_);

  // Grab params off the param server
  private_nh_.param("use_map_topic", use_map_topic_, false);
  private_nh_.param("first_map_only", first_map_only_, false);

  double tmp;
  private_nh_.param("gui_publish_rate", tmp, -1.0);
  gui_publish_period = ros::Duration(1.0/tmp);
  private_nh_.param("save_pose_rate", tmp, 0.5);
  save_pose_period = ros::Duration(1.0/tmp);
```

```
private_nh_.param("laser_min_range", laser_min_range_, -1.0);
private_nh_.param("laser_max_range", laser_max_range_, -1.0);
private_nh_.param("laser_max_beams", max_beams_, 30);
private_nh_.param("min_particles", min_particles_, 100);
private_nh_.param("max_particles", max_particles_, 5000);
private_nh_.param("kld_err", pf_err_, 0.01);
private_nh_.param("kld_z", pf_z_, 0.99);
private_nh_.param("odom_alpha1", alpha1_, 0.2);
private_nh_.param("odom_alpha2", alpha2_, 0.2);
private_nh_.param("odom_alpha3", alpha3_, 0.2);
private_nh_.param("odom_alpha4", alpha4_, 0.2);
private_nh_.param("odom_alpha5", alpha5_, 0.2);

private_nh_.param("do_beamskip", do_beamskip_, false);
private_nh_.param("beam_skip_distance", beam_skip_distance_, 0.5);
private_nh_.param("beam_skip_threshold", beam_skip_threshold_, 0.3);
private_nh_.param("beam_skip_error_threshold_", beam_skip_error_threshold_, 0.9);

private_nh_.param("laser_z_hit", z_hit_, 0.95);
private_nh_.param("laser_z_short", z_short_, 0.1);
private_nh_.param("laser_z_max", z_max_, 0.05);
private_nh_.param("laser_z_rand", z_rand_, 0.05);
private_nh_.param("laser_sigma_hit", sigma_hit_, 0.2);
private_nh_.param("laser_lambda_short", lambda_short_, 0.1);
private_nh_.param("laser_likelihood_max_dist", laser_likelihood_max_dist_, 2.0);
std::string tmp_model_type;
private_nh_.param("laser_model_type", tmp_model_type, std::string("likelihood_field"));
if(tmp_model_type == "beam")
  laser_model_type_ = LASER_MODEL_BEAM;
else if(tmp_model_type == "likelihood_field")
  laser_model_type_ = LASER_MODEL_LIKELIHOOD_FIELD;
else if(tmp_model_type == "likelihood_field_prob"){
  laser_model_type_ = LASER_MODEL_LIKELIHOOD_FIELD_PROB;
}
else
{
  ROS_WARN("Unknown laser model type \"%s\"; defaulting to likelihood_field model",
           tmp_model_type.c_str());
  laser_model_type_ = LASER_MODEL_LIKELIHOOD_FIELD;
}

private_nh_.param("odom_model_type", tmp_model_type, std::string("diff"));
if(tmp_model_type == "diff")
  odom_model_type_ = ODOM_MODEL_DIFF;
else if(tmp_model_type == "omni")
```

```
    odom_model_type_ = ODOM_MODEL_OMNI;
  else if(tmp_model_type == "diff-corrected")
    odom_model_type_ = ODOM_MODEL_DIFF_CORRECTED;
  else if(tmp_model_type == "omni-corrected")
    odom_model_type_ = ODOM_MODEL_OMNI_CORRECTED;
  else
  {
    ROS_WARN("Unknown odom model type \"%s\"; defaulting to diff model",
             tmp_model_type.c_str());
    odom_model_type_ = ODOM_MODEL_DIFF;
  }

  private_nh_.param("update_min_d", d_thresh_, 0.2);
  private_nh_.param("update_min_a", a_thresh_, M_PI/6.0);
  private_nh_.param("odom_frame_id", odom_frame_id_, std::string("odom"));
  private_nh_.param("base_frame_id", base_frame_id_, std::string("base_link"));
  private_nh_.param("global_frame_id", global_frame_id_, std::string("map"));
  private_nh_.param("resample_interval", resample_interval_, 2);
  double tmp_tol;
  private_nh_.param("transform_tolerance", tmp_tol, 0.1);
  private_nh_.param("recovery_alpha_slow", alpha_slow_, 0.001);
  private_nh_.param("recovery_alpha_fast", alpha_fast_, 0.1);
  private_nh_.param("tf_broadcast", tf_broadcast_, true);

  transform_tolerance_.fromSec(tmp_tol);

  {
    double bag_scan_period;
    private_nh_.param("bag_scan_period", bag_scan_period, -1.0);
    bag_scan_period_.fromSec(bag_scan_period);
  }

  updatePoseFromServer();

  cloud_pub_interval.fromSec(1.0);
  tfb_ = new tf::TransformBroadcaster();
  tf_ = new TransformListenerWrapper();

  pose_pub_ = nh_.advertise<geometry_msgs::PoseWithCovarianceStamped>("amcl_pose", 2,
true);
  particlecloud_pub_ = nh_.advertise<geometry_msgs::PoseArray>("particlecloud", 2, true);
  particlecloud_pub_PreResample =
nh_.advertise<geometry_msgs::PoseArray>("particlecloudPreResample", 2, true);


//Set up custom message publishers
```

```
test_pub_ = nh_.advertise<testmsg>("test_pub", 2, true);
numsamples_pub_ = nh_.advertise<numsamples_msg>("numsamples", 2, true);
max_weight_hyp_pub_ = nh_.advertise<max_weight_hyp_msg>("max_weight_hyp", 2, true);
cluster_count_pub_ = nh_.advertise<cluster_count_msg>("cluster_count", 2, true);
delta_pose_pub_ = nh_.advertise<delta_pose_msg>("delta_pose", 2, true);
filter_covariance_pub_ = nh_.advertise<filter_covariance_msg>("filter_covariance", 2, true);

cloud_weight_pub_ = nh_.advertise<filter_covariance_msg>("cloud_weight",2,true);

cloud_weight_pub_beforeUpdateSensor =
nh_.advertise<filter_covariance_msg>("cloud_weight_beforeUpdateSensor",2,true);//cloud_wei
ght_pub_ = nh_.advertise<cloud_weight_msg>("cloud_weight",2,true);
//cloud_weight_pub_ = nh_.advertise<std_msgs::Int32MultiArray>("cloud_weight",2, true);

  global_loc_srv_ = nh_.advertiseService("global_localization",
                                         &AmclNode::globalLocalizationCallback,
                                 this);
  nomotion_update_srv_= nh_.advertiseService("request_nomotion_update",
&AmclNode::nomotionUpdateCallback, this);
  set_map_srv_= nh_.advertiseService("set_map", &AmclNode::setMapCallback, this);

  laser_scan_sub_ = new message_filters::Subscriber<sensor_msgs::LaserScan>(nh_,
scan_topic_, 100);
  laser_scan_filter_ =
        new tf::MessageFilter<sensor_msgs::LaserScan>(*laser_scan_sub_,
                                           *tf_,
                                           odom_frame_id_,
                                           100);
  laser_scan_filter_->registerCallback(boost::bind(&AmclNode::laserReceived,
                                       this, _1));
  initial_pose_sub_ = nh_.subscribe("initialpose", 2, &AmclNode::initialPoseReceived, this);

  if(use_map_topic_) {
    map_sub_ = nh_.subscribe("map", 1, &AmclNode::mapReceived, this);
    ROS_INFO("Subscribed to map topic.");
  } else {
    requestMap();
  }
  m_force_update = false;

  dsrv_ = new dynamic_reconfigure::Server<amcl::AMCLConfig>(ros::NodeHandle("~"));
  dynamic_reconfigure::Server<amcl::AMCLConfig>::CallbackType cb =
boost::bind(&AmclNode::reconfigureCB, this, _1, _2);
  dsrv_->setCallback(cb);

  // 15s timer to warn on lack of receipt of laser scans, #5209
```

```
  laser_check_interval_ = ros::Duration(15.0);
  check_laser_timer_ = nh_.createTimer(laser_check_interval_,
                                       boost::bind(&AmclNode::checkLaserReceived, this, _1));
}

void AmclNode::reconfigureCB(AMCLConfig &config, uint32_t level)
{
  boost::recursive_mutex::scoped_lock cfl(configuration_mutex_);

  //we don't want to do anything on the first call
  //which corresponds to startup
  if(first_reconfigure_call_)
  {
    first_reconfigure_call_ = false;
    default_config_ = config;
    return;
  }

  if(config.restore_defaults) {
    config = default_config_;
    //avoid looping
    config.restore_defaults = false;
  }

  d_thresh_ = config.update_min_d;
  a_thresh_ = config.update_min_a;

  resample_interval_ = config.resample_interval;

  laser_min_range_ = config.laser_min_range;
  laser_max_range_ = config.laser_max_range;

  gui_publish_period = ros::Duration(1.0/config.gui_publish_rate);
  save_pose_period = ros::Duration(1.0/config.save_pose_rate);

  transform_tolerance_.fromSec(config.transform_tolerance);

  max_beams_ = config.laser_max_beams;
  alpha1_ = config.odom_alpha1;
  alpha2_ = config.odom_alpha2;
  alpha3_ = config.odom_alpha3;
  alpha4_ = config.odom_alpha4;
  alpha5_ = config.odom_alpha5;

  z_hit_ = config.laser_z_hit;
  z_short_ = config.laser_z_short;
```

```
  z_max_ = config.laser_z_max;
  z_rand_ = config.laser_z_rand;
  sigma_hit_ = config.laser_sigma_hit;
  lambda_short_ = config.laser_lambda_short;
  laser_likelihood_max_dist_ = config.laser_likelihood_max_dist;

  if(config.laser_model_type == "beam")
    laser_model_type_ = LASER_MODEL_BEAM;
  else if(config.laser_model_type == "likelihood_field")
    laser_model_type_ = LASER_MODEL_LIKELIHOOD_FIELD;
  else if(config.laser_model_type == "likelihood_field_prob")
    laser_model_type_ = LASER_MODEL_LIKELIHOOD_FIELD_PROB;

  if(config.odom_model_type == "diff")
    odom_model_type_ = ODOM_MODEL_DIFF;
  else if(config.odom_model_type == "omni")
    odom_model_type_ = ODOM_MODEL_OMNI;
  else if(config.odom_model_type == "diff-corrected")
    odom_model_type_ = ODOM_MODEL_DIFF_CORRECTED;
  else if(config.odom_model_type == "omni-corrected")
    odom_model_type_ = ODOM_MODEL_OMNI_CORRECTED;

  if(config.min_particles > config.max_particles)
  {
    ROS_WARN("You've set min_particles to be greater than max particles, this isn't allowed so
they'll be set to be equal.");
    config.max_particles = config.min_particles;
  }

  min_particles_ = config.min_particles;
  max_particles_ = config.max_particles;
  alpha_slow_ = config.recovery_alpha_slow;
  alpha_fast_ = config.recovery_alpha_fast;
  tf_broadcast_ = config.tf_broadcast;

  do_beamskip_= config.do_beamskip;
  beam_skip_distance_ = config.beam_skip_distance;
  beam_skip_threshold_ = config.beam_skip_threshold;

  pf_ = pf_alloc(min_particles_, max_particles_,
                 alpha_slow_, alpha_fast_,
                 (pf_init_model_fn_t)AmclNode::uniformPoseGenerator,
                 (void *)map_);
  pf_err_ = config.kld_err;
  pf_z_ = config.kld_z;
  pf_->pop_err = pf_err_;
```

```
pf_->pop_z = pf_z_;

// Initialize the filter
pf_vector_t pf_init_pose_mean = pf_vector_zero();
pf_init_pose_mean.v[0] = last_published_pose.pose.pose.position.x;
pf_init_pose_mean.v[1] = last_published_pose.pose.pose.position.y;
pf_init_pose_mean.v[2] = tf::getYaw(last_published_pose.pose.pose.orientation);
pf_matrix_t pf_init_pose_cov = pf_matrix_zero();
pf_init_pose_cov.m[0][0] = last_published_pose.pose.covariance[6*0+0];
pf_init_pose_cov.m[1][1] = last_published_pose.pose.covariance[6*1+1];
pf_init_pose_cov.m[2][2] = last_published_pose.pose.covariance[6*5+5];
pf_init(pf_, pf_init_pose_mean, pf_init_pose_cov);
pf_init_ = false;

// Instantiate the sensor objects
// Odometry
delete odom_;
odom_ = new AMCLOdom();
ROS_ASSERT(odom_);
odom_->SetModel( odom_model_type_, alpha1_, alpha2_, alpha3_, alpha4_, alpha5_ );
// Laser
delete laser_;
laser_ = new AMCLLaser(max_beams_, map_);
ROS_ASSERT(laser_);
if(laser_model_type_ == LASER_MODEL_BEAM)
  laser_->SetModelBeam(z_hit_, z_short_, z_max_, z_rand_,
                       sigma_hit_, lambda_short_, 0.0);
else if(laser_model_type_ == LASER_MODEL_LIKELIHOOD_FIELD_PROB){
  ROS_INFO("Initializing likelihood field model; this can take some time on large maps...");
  laser_->SetModelLikelihoodFieldProb(z_hit_, z_rand_, sigma_hit_,
                                      laser_likelihood_max_dist_,
                                      do_beamskip_, beam_skip_distance_,
                                      beam_skip_threshold_, beam_skip_error_threshold_);
  ROS_INFO("Done initializing likelihood field model with probabilities.");
}
else if(laser_model_type_ == LASER_MODEL_LIKELIHOOD_FIELD){
  ROS_INFO("Initializing likelihood field model; this can take some time on large maps...");
  laser_->SetModelLikelihoodField(z_hit_, z_rand_, sigma_hit_,
                                  laser_likelihood_max_dist_);
  ROS_INFO("Done initializing likelihood field model.");
}

odom_frame_id_ = config.odom_frame_id;
base_frame_id_ = config.base_frame_id;
global_frame_id_ = config.global_frame_id;
```

```
  delete laser_scan_filter_;
  laser_scan_filter_ =
        new tf::MessageFilter<sensor_msgs::LaserScan>(*laser_scan_sub_,
                                                      *tf_,
                                                      odom_frame_id_,
                                                      100);
  laser_scan_filter_->registerCallback(boost::bind(&AmclNode::laserReceived,
                                       this, _1));

  initial_pose_sub_ = nh_.subscribe("initialpose", 2, &AmclNode::initialPoseReceived, this);
}


void AmclNode::runFromBag(const std::string &in_bag_fn)
{
  rosbag::Bag bag;
  bag.open(in_bag_fn, rosbag::bagmode::Read);
  std::vector<std::string> topics;
  topics.push_back(std::string("tf"));
  std::string scan_topic_name = "base_scan"; // TODO determine what topic this actually is from
ROS
  topics.push_back(scan_topic_name);
  rosbag::View view(bag, rosbag::TopicQuery(topics));

  ros::Publisher laser_pub = nh_.advertise<sensor_msgs::LaserScan>(scan_topic_name, 100);
  ros::Publisher tf_pub = nh_.advertise<tf2_msgs::TFMessage>("/tf", 100);

  // Sleep for a second to let all subscribers connect
  ros::WallDuration(1.0).sleep();

  ros::WallTime start(ros::WallTime::now());

  // Wait for map
  while (ros::ok())
  {
    {
      boost::recursive_mutex::scoped_lock cfl(configuration_mutex_);
      if (map_)
      {
        ROS_INFO("Map is ready");
        break;
      }
    }
    ROS_INFO("Waiting for map...");
    ros::getGlobalCallbackQueue()->callAvailable(ros::WallDuration(1.0));
  }
```

```
BOOST_FOREACH(rosbag::MessageInstance const msg, view)
{
  if (!ros::ok())
  {
    break;
  }

  // Process any ros messages or callbacks at this point
  ros::getGlobalCallbackQueue()->callAvailable(ros::WallDuration());

  tf2_msgs::TFMessage::ConstPtr tf_msg = msg.instantiate<tf2_msgs::TFMessage>();
  if (tf_msg != NULL)
  {
    tf_pub.publish(msg);
    for (size_t ii=0; ii<tf_msg->transforms.size(); ++ii)
    {
      tf_->getBuffer().setTransform(tf_msg->transforms[ii], "rosbag_authority");
    }
    continue;
  }

  sensor_msgs::LaserScan::ConstPtr base_scan = msg.instantiate<sensor_msgs::LaserScan>();
  if (base_scan != NULL)
  {
    laser_pub.publish(msg);
    laser_scan_filter_->add(base_scan);
    if (bag_scan_period_ > ros::WallDuration(0))
    {
      bag_scan_period_.sleep();
    }
    continue;
  }

  ROS_WARN_STREAM("Unsupported message type" << msg.getTopic());
}

bag.close();

double runtime = (ros::WallTime::now() - start).toSec();
ROS_INFO("Bag complete, took %.1f seconds to process, shutting down", runtime);

const geometry_msgs::Quaternion & q(last_published_pose.pose.pose.orientation);
double yaw, pitch, roll;
tf::Matrix3x3(tf::Quaternion(q.x, q.y, q.z, q.w)).getEulerYPR(yaw,pitch,roll);
ROS_INFO("Final location %.3f, %.3f, %.3f with stamp=%f",
```

```
          last_published_pose.pose.pose.position.x,
          last_published_pose.pose.pose.position.y,
          yaw, last_published_pose.header.stamp.toSec()
          );

  ros::shutdown();
}


void AmclNode::savePoseToServer()
{

  // We need to apply the last transform to the latest odom pose to get
  // the latest map pose to store.  We'll take the covariance from
  // last_published_pose.
  tf::Pose map_pose = latest_tf_.inverse() * latest_odom_pose_;
  double yaw,pitch,roll;
  map_pose.getBasis().getEulerYPR(yaw, pitch, roll);

  ROS_DEBUG("Saving pose to server. x: %.3f, y: %.3f", map_pose.getOrigin().x(),
map_pose.getOrigin().y() );

  private_nh_.setParam("initial_pose_x", map_pose.getOrigin().x());
  private_nh_.setParam("initial_pose_y", map_pose.getOrigin().y());
  private_nh_.setParam("initial_pose_a", yaw);
  private_nh_.setParam("initial_cov_xx",
                       last_published_pose.pose.covariance[6*0+0]);
  private_nh_.setParam("initial_cov_yy",
                       last_published_pose.pose.covariance[6*1+1]);
  private_nh_.setParam("initial_cov_aa",
                       last_published_pose.pose.covariance[6*5+5]);
}

void AmclNode::updatePoseFromServer()
{
  init_pose_[0] = 0.0;
  init_pose_[1] = 0.0;
  init_pose_[2] = 0.0;
  init_cov_[0] = 0.5 * 0.5;
  init_cov_[1] = 0.5 * 0.5;
  init_cov_[2] = (M_PI/12.0) * (M_PI/12.0);
  // Check for NAN on input from param server, #5239
  double tmp_pos;
  private_nh_.param("initial_pose_x", tmp_pos, init_pose_[0]);
  if(!std::isnan(tmp_pos))
    init_pose_[0] = tmp_pos;
```

```
  else
    ROS_WARN("ignoring NAN in initial pose X position");
  private_nh_.param("initial_pose_y", tmp_pos, init_pose_[1]);
  if(!std::isnan(tmp_pos))
    init_pose_[1] = tmp_pos;
  else
    ROS_WARN("ignoring NAN in initial pose Y position");
  private_nh_.param("initial_pose_a", tmp_pos, init_pose_[2]);
  if(!std::isnan(tmp_pos))
    init_pose_[2] = tmp_pos;
  else
    ROS_WARN("ignoring NAN in initial pose Yaw");
  private_nh_.param("initial_cov_xx", tmp_pos, init_cov_[0]);
  if(!std::isnan(tmp_pos))
    init_cov_[0] =tmp_pos;
  else
    ROS_WARN("ignoring NAN in initial covariance XX");
  private_nh_.param("initial_cov_yy", tmp_pos, init_cov_[1]);
  if(!std::isnan(tmp_pos))
    init_cov_[1] = tmp_pos;
  else
    ROS_WARN("ignoring NAN in initial covariance YY");
  private_nh_.param("initial_cov_aa", tmp_pos, init_cov_[2]);
  if(!std::isnan(tmp_pos))
    init_cov_[2] = tmp_pos;
  else
    ROS_WARN("ignoring NAN in initial covariance AA");
}

void
AmclNode::checkLaserReceived(const ros::TimerEvent& event)
{
  ros::Duration d = ros::Time::now() - last_laser_received_ts_;
  if(d > laser_check_interval_)
  {
    ROS_WARN("No laser scan received (and thus no pose updates have been published) for %f
seconds.  Verify that data is being published on the %s topic.",
             d.toSec(),
             ros::names::resolve(scan_topic_).c_str());
  }
}

void
AmclNode::requestMap()
{
  boost::recursive_mutex::scoped_lock ml(configuration_mutex_);
```

```
  // get map via RPC
  nav_msgs::GetMap::Request  req;
  nav_msgs::GetMap::Response resp;
  ROS_INFO("Requesting the map...");
  while(!ros::service::call("static_map", req, resp))
  {
    ROS_WARN("Request for map failed; trying again...");
    ros::Duration d(0.5);
    d.sleep();
  }
  handleMapMessage( resp.map );
}

void
AmclNode::mapReceived(const nav_msgs::OccupancyGridConstPtr& msg)
{
  if( first_map_only_ && first_map_received_ ) {
    return;
  }

  handleMapMessage( *msg );

  first_map_received_ = true;
}

void
AmclNode::handleMapMessage(const nav_msgs::OccupancyGrid& msg)
{
  boost::recursive_mutex::scoped_lock cfl(configuration_mutex_);

  ROS_INFO("Received a %d X %d map @ %.3f m/pix\n",
           msg.info.width,
           msg.info.height,
           msg.info.resolution);

  freeMapDependentMemory();
  // Clear queued laser objects because they hold pointers to the existing
  // map, #5202.
  lasers_.clear();
  lasers_update_.clear();
  frame_to_laser_.clear();

  map_ = convertMap(msg);

#if NEW_UNIFORM_SAMPLING
```

```
  // Index of free space
  free_space_indices.resize(0);
  for(int i = 0; i < map_->size_x; i++)
    for(int j = 0; j < map_->size_y; j++)
      if(map_->cells[MAP_INDEX(map_,i,j)].occ_state == -1)
        free_space_indices.push_back(std::make_pair(i,j));
#endif
ROS_INFO("calling pf_alloc here");
  // Create the particle filter
  pf_ = pf_alloc(min_particles_, max_particles_,
                 alpha_slow_, alpha_fast_,
                 (pf_init_model_fn_t)AmclNode::uniformPoseGenerator,
                 (void *)map_);
  pf_->pop_err = pf_err_;
  pf_->pop_z = pf_z_;

  // Initialize the filter
  updatePoseFromServer();
  pf_vector_t pf_init_pose_mean = pf_vector_zero();
  pf_init_pose_mean.v[0] = init_pose_[0];
  pf_init_pose_mean.v[1] = init_pose_[1];
  pf_init_pose_mean.v[2] = init_pose_[2];
  pf_matrix_t pf_init_pose_cov = pf_matrix_zero();
  pf_init_pose_cov.m[0][0] = init_cov_[0];
  pf_init_pose_cov.m[1][1] = init_cov_[1];
  pf_init_pose_cov.m[2][2] = init_cov_[2];
  pf_init(pf_, pf_init_pose_mean, pf_init_pose_cov);
  pf_init_ = false;

  // Instantiate the sensor objects
  // Odometry
  delete odom_;
  odom_ = new AMCLOdom();
  ROS_ASSERT(odom_);
  odom_->SetModel( odom_model_type_, alpha1_, alpha2_, alpha3_, alpha4_, alpha5_ );
  // Laser
  delete laser_;
  laser_ = new AMCLLaser(max_beams_, map_);
  ROS_ASSERT(laser_);
  if(laser_model_type_ == LASER_MODEL_BEAM)
    laser_->SetModelBeam(z_hit_, z_short_, z_max_, z_rand_,
                         sigma_hit_, lambda_short_, 0.0);
  else if(laser_model_type_ == LASER_MODEL_LIKELIHOOD_FIELD_PROB){
    ROS_INFO("Initializing likelihood field model; this can take some time on large maps...");
    laser_->SetModelLikelihoodFieldProb(z_hit_, z_rand_, sigma_hit_,
                                        laser_likelihood_max_dist_,
```

```
                                       do_beamskip_, beam_skip_distance_,
                                       beam_skip_threshold_, beam_skip_error_threshold_);
    ROS_INFO("Done initializing likelihood field model.");
  }
  else
  {
    ROS_INFO("Initializing likelihood field model; this can take some time on large maps...");
    laser_->SetModelLikelihoodField(z_hit_, z_rand_, sigma_hit_,
                                    laser_likelihood_max_dist_);
    ROS_INFO("Done initializing likelihood field model.");
  }

  // In case the initial pose message arrived before the first map,
  // try to apply the initial pose now that the map has arrived.
  applyInitialPose();

}

void
AmclNode::freeMapDependentMemory()
{
  if( map_ != NULL ) {
    map_free( map_ );
    map_ = NULL;
  }
  if( pf_ != NULL ) {
    pf_free( pf_ );
    pf_ = NULL;
  }
  delete odom_;
  odom_ = NULL;
  delete laser_;
  laser_ = NULL;
}

/**
 * Convert an OccupancyGrid map message into the internal
 * representation.  This allocates a map_t and returns it.
 */
map_t*
AmclNode::convertMap( const nav_msgs::OccupancyGrid& map_msg )
{
  map_t* map = map_alloc();
  ROS_ASSERT(map);

  map->size_x = map_msg.info.width;
```

```
  map->size_y = map_msg.info.height;
  map->scale = map_msg.info.resolution;
  map->origin_x = map_msg.info.origin.position.x + (map->size_x / 2) * map->scale;
  map->origin_y = map_msg.info.origin.position.y + (map->size_y / 2) * map->scale;
  // Convert to player format
  map->cells = (map_cell_t*)malloc(sizeof(map_cell_t)*map->size_x*map->size_y);
  ROS_ASSERT(map->cells);
  for(int i=0;i<map->size_x * map->size_y;i++)
  {
    if(map_msg.data[i] == 0)
      map->cells[i].occ_state = -1;
    else if(map_msg.data[i] == 100)
      map->cells[i].occ_state = +1;
    else
      map->cells[i].occ_state = 0;
  }

  return map;
}

AmclNode::~AmclNode()
{
  delete dsrv_;
  freeMapDependentMemory();
  delete laser_scan_filter_;
  delete laser_scan_sub_;
  delete tfb_;
  delete tf_;
  // TODO: delete everything allocated in constructor
}

bool
AmclNode::getOdomPose(tf::Stamped<tf::Pose>& odom_pose,
                      double& x, double& y, double& yaw,
                      const ros::Time& t, const std::string& f)
{
  // Get the robot's pose
  tf::Stamped<tf::Pose> ident (tf::Transform(tf::createIdentityQuaternion(),
                                             tf::Vector3(0,0,0)), t, f);
  try
  {
    this->tf_->transformPose(odom_frame_id_, ident, odom_pose);
  }
  catch(tf::TransformException e)
  {
    ROS_WARN("Failed to compute odom pose, skipping scan (%s)", e.what());
```

```
    return false;
  }
  x = odom_pose.getOrigin().x();
  y = odom_pose.getOrigin().y();
  double pitch,roll;
  odom_pose.getBasis().getEulerYPR(yaw, pitch, roll);

  return true;
}


pf_vector_t
AmclNode::uniformPoseGenerator(void* arg)
{
  map_t* map = (map_t*)arg;
#if NEW_UNIFORM_SAMPLING
  unsigned int rand_index = drand48() * free_space_indices.size();
  std::pair<int,int> free_point = free_space_indices[rand_index];
  pf_vector_t p;
  p.v[0] = MAP_WXGX(map, free_point.first);
  p.v[1] = MAP_WYGY(map, free_point.second);
  p.v[2] = drand48() * 2 * M_PI - M_PI;
#else
  double min_x, max_x, min_y, max_y;

  min_x = (map->size_x * map->scale)/2.0 - map->origin_x;
  max_x = (map->size_x * map->scale)/2.0 + map->origin_x;
  min_y = (map->size_y * map->scale)/2.0 - map->origin_y;
  max_y = (map->size_y * map->scale)/2.0 + map->origin_y;

  pf_vector_t p;

  ROS_DEBUG("Generating new uniform sample");
  for(;;)
  {
    p.v[0] = min_x + drand48() * (max_x - min_x);
    p.v[1] = min_y + drand48() * (max_y - min_y);
    p.v[2] = drand48() * 2 * M_PI - M_PI;
    // Check that it's a free cell
    int i,j;
    i = MAP_GXWX(map, p.v[0]);
    j = MAP_GYWY(map, p.v[1]);
    if(MAP_VALID(map,i,j) && (map->cells[MAP_INDEX(map,i,j)].occ_state == -1))
      break;
  }
#endif
```

```
  return p;
}

bool
AmclNode::globalLocalizationCallback(std_srvs::Empty::Request& req,
                                     std_srvs::Empty::Response& res)
{
  if( map_ == NULL ) {
    return true;
  }
  boost::recursive_mutex::scoped_lock gl(configuration_mutex_);
  ROS_INFO("Initializing with uniform distribution");
  pf_init_model(pf_, (pf_init_model_fn_t)AmclNode::uniformPoseGenerator,
                (void *)map_);
  ROS_INFO("Global initialisation done!");
  pf_init_ = false;
  return true;
}

// force nomotion updates (amcl updating without requiring motion)
bool
AmclNode::nomotionUpdateCallback(std_srvs::Empty::Request& req,
                                 std_srvs::Empty::Response& res)
{
        m_force_update = true;
        //ROS_INFO("Requesting no-motion update");
        return true;
}

bool
AmclNode::setMapCallback(nav_msgs::SetMap::Request& req,
                         nav_msgs::SetMap::Response& res)
{
  handleMapMessage(req.map);
  handleInitialPoseMessage(req.initial_pose);
  res.success = true;
  return true;
}

void
AmclNode::laserReceived(const sensor_msgs::LaserScanConstPtr& laser_scan)
{
  last_laser_received_ts_ = ros::Time::now();
  if( map_ == NULL ) {
    return;
  }
```

```
  boost::recursive_mutex::scoped_lock lr(configuration_mutex_);
  int laser_index = -1;

  // Do we have the base->base_laser Tx yet?
  if(frame_to_laser_.find(laser_scan->header.frame_id) == frame_to_laser_.end())
  {
    ROS_DEBUG("Setting up laser %d (frame_id=%s)\n", (int)frame_to_laser_.size(),
laser_scan->header.frame_id.c_str());
    lasers_.push_back(new AMCLLaser(*laser_));
    lasers_update_.push_back(true);
    laser_index = frame_to_laser_.size();

    tf::Stamped<tf::Pose> ident (tf::Transform(tf::createIdentityQuaternion(),
                                       tf::Vector3(0,0,0)),
                                 ros::Time(), laser_scan->header.frame_id);
    tf::Stamped<tf::Pose> laser_pose;
    try
    {
      this->tf_->transformPose(base_frame_id_, ident, laser_pose);
    }
    catch(tf::TransformException& e)
    {
      ROS_ERROR("Couldn't transform from %s to %s, "
                "even though the message notifier is in use",
                laser_scan->header.frame_id.c_str(),
                base_frame_id_.c_str());
      return;
    }

    pf_vector_t laser_pose_v;
    laser_pose_v.v[0] = laser_pose.getOrigin().x();
    laser_pose_v.v[1] = laser_pose.getOrigin().y();
    // laser mounting angle gets computed later -> set to 0 here!
    laser_pose_v.v[2] = 0;
    lasers_[laser_index]->SetLaserPose(laser_pose_v);
    ROS_DEBUG("Received laser's pose wrt robot: %.3f %.3f %.3f",
              laser_pose_v.v[0],
              laser_pose_v.v[1],
              laser_pose_v.v[2]);

    frame_to_laser_[laser_scan->header.frame_id] = laser_index;
  } else {
    // we have the laser pose, retrieve laser index
    laser_index = frame_to_laser_[laser_scan->header.frame_id];
  }
```

```
// Where was the robot when this scan was taken?
pf_vector_t pose;
if(!getOdomPose(latest_odom_pose_, pose.v[0], pose.v[1], pose.v[2],
                laser_scan->header.stamp, base_frame_id_))
{
  ROS_ERROR("Couldn't determine robot's pose associated with laser scan");
  return;
}


pf_vector_t delta = pf_vector_zero();

if(pf_init_)
{
  // Compute change in pose
  //delta = pf_vector_coord_sub(pose, pf_odom_pose_);
  delta.v[0] = pose.v[0] - pf_odom_pose_.v[0];
  delta.v[1] = pose.v[1] - pf_odom_pose_.v[1];
  delta.v[2] = angle_diff(pose.v[2], pf_odom_pose_.v[2]);

  //Publish the deltas
  delta_pose_msg dpm;
  dpm.v1 = delta.v[0];
  dpm.v2 = delta.v[1];
  dpm.v3 = delta.v[2];
  delta_pose_pub_.publish(dpm);


  // See if we should update the filter
  bool update = fabs(delta.v[0]) > d_thresh_ ||
                fabs(delta.v[1]) > d_thresh_ ||
                fabs(delta.v[2]) > a_thresh_;
  update = update || m_force_update;
  m_force_update=false;

 if(update)
    ROS_INFO("Update:TRUE");


  // Set the laser update flags
  if(update)
    for(unsigned int i=0; i < lasers_update_.size(); i++)
      lasers_update_[i] = true;
}

bool force_publication = false;
```

```
if(!pf_init_)
{
        ROS_INFO("Ln 1174, !pf_init. Updating sensor data");
  // Pose at last filter update
  pf_odom_pose_ = pose;

  // Filter is now initialized
  pf_init_ = true;

  // Should update sensor data
  for(unsigned int i=0; i < lasers_update_.size(); i++)
    lasers_update_[i] = true;

  force_publication = true;

  resample_count_ = 0;
}
// If the robot has moved, update the filter
else if(pf_init_ && lasers_update_[laser_index])
{
  //printf("pose\n");
  //pf_vector_fprintf(pose, stdout, "%.3f");

  AMCLOdomData odata;
  odata.pose = pose;
  // HACK
  // Modify the delta in the action data so the filter gets
  // updated correctly
  odata.delta = delta;

  ROS_INFO("Robot has moved. updating filter. Line 1202.");
  // Use the action data to update the filter
  odom_->UpdateAction(pf_, (AMCLSensorData*)&odata);

  // Pose at last filter update
  //this->pf_odom_pose = pose;
}

bool resampled = false;
// If the robot has moved, update the filter
if(lasers_update_[laser_index])
{
  ROS_INFO("Robot has moved. updating filter. Line 1214.");
  AMCLLaserData ldata;
  ldata.sensor = lasers_[laser_index];
  ldata.range_count = laser_scan->ranges.size();
```

```
  // To account for lasers that are mounted upside-down, we determine the
  // min, max, and increment angles of the laser in the base frame.
  //
  // Construct min and max angles of laser, in the base_link frame.
  tf::Quaternion q;
  q.setRPY(0.0, 0.0, laser_scan->angle_min);
  tf::Stamped<tf::Quaternion> min_q(q, laser_scan->header.stamp,
                                    laser_scan->header.frame_id);
  q.setRPY(0.0, 0.0, laser_scan->angle_min + laser_scan->angle_increment);
  tf::Stamped<tf::Quaternion> inc_q(q, laser_scan->header.stamp,
                                    laser_scan->header.frame_id);
  try
  {
    tf_->transformQuaternion(base_frame_id_, min_q, min_q);
    tf_->transformQuaternion(base_frame_id_, inc_q, inc_q);
  }
  catch(tf::TransformException& e)
  {
    ROS_WARN("Unable to transform min/max laser angles into base frame: %s",
             e.what());
    return;
  }

  double angle_min = tf::getYaw(min_q);
  double angle_increment = tf::getYaw(inc_q) - angle_min;

  // wrapping angle to [-pi .. pi]
  angle_increment = fmod(angle_increment + 5*M_PI, 2*M_PI) - M_PI;

  ROS_DEBUG("Laser %d angles in base frame: min: %.3f inc: %.3f", laser_index, angle_min,
angle_increment);

  // Apply range min/max thresholds, if the user supplied them
  if(laser_max_range_ > 0.0)
    ldata.range_max = std::min(laser_scan->range_max, (float)laser_max_range_);
  else
    ldata.range_max = laser_scan->range_max;
  double range_min;
  if(laser_min_range_ > 0.0)
    range_min = std::max(laser_scan->range_min, (float)laser_min_range_);
  else
    range_min = laser_scan->range_min;
  // The AMCLLaserData destructor will free this memory
  ldata.ranges = new double[ldata.range_count][2];
  ROS_ASSERT(ldata.ranges);
```

```
   for(int i=0;i<ldata.range_count;i++)
   {
     // amcl doesn't (yet) have a concept of min range.  So we'll map short
     // readings to max range.
     if(laser_scan->ranges[i] <= range_min)
       ldata.ranges[i][0] = ldata.range_max;
     else
       ldata.ranges[i][0] = laser_scan->ranges[i];
     // Compute bearing
     ldata.ranges[i][1] = angle_min +
           (i * angle_increment);
   }



//The particle filter weights are uniformly-distributed prior to UpdateSensor
   lasers_[laser_index]->UpdateSensor(pf_, (AMCLSensorData*)&ldata);

   lasers_update_[laser_index] = false;

   pf_odom_pose_ = pose;

//BEGIN: This line is where the guy from the forum says to try printing the particle weights
          pf_sample_set_t* set_temp = pf_->sets + pf_->current_set;
          filter_covariance_msg weights_msg;
          std::vector<float> weights(set_temp->sample_count);


          for(int i=0;i<set_temp->sample_count;i++)
               {
                    weights[i] = set_temp->samples[i].weight;
             ROS_DEBUG("Time:%f,Sample: %i, PoseX:%f, PoseY:%f, Weight:
%f",ros::Time::now().toSec(),i, set_temp->samples[i].pose.v[0],set_temp-
>samples[i].pose.v[1],set_temp->samples[i].weight);
          }
          weights_msg.cov = weights;
    cloud_weight_pub_.publish(weights_msg);

//Print the poses that corresponde. Might the same as particlecloud topic, but just in case...
  geometry_msgs::PoseArray cloud_msg_PreResample;


     cloud_msg_PreResample.header.stamp = ros::Time::now();
     cloud_msg_PreResample.header.frame_id = global_frame_id_;
     cloud_msg_PreResample.poses.resize(set_temp->sample_count);
     for(int i=0;i<set_temp->sample_count;i++)
```

```
    {
      tf::poseTFToMsg(tf::Pose(tf::createQuaternionFromYaw(set_temp->samples[i].pose.v[2]),
                               tf::Vector3(set_temp->samples[i].pose.v[0],
                                           set_temp->samples[i].pose.v[1], 0)),
                      cloud_msg_PreResample.poses[i]);



    }
    particlecloud_pub_PreResample.publish(cloud_msg_PreResample);



//END: This line is where the guy from the forum says to try printing the particle weights



   // Resample the particles
   if(!(++resample_count_ % resample_interval_))
   {
     ROS_INFO("Resample Count= %d",resample_count_);
     ROS_INFO("Resample Interval= %d",resample_interval_);
          ROS_INFO("Resample Count mod Interval = %d", resample_count_ %
resample_interval_);
          ROS_INFO("Updating Particle Filter. Line 1331.");
      pf_update_resample(pf_);
     resampled = true;


   }
/*I am trying to track down the part of the code that changes the weight ddistribution*/
          pf_sample_set_t* set_test = pf_->sets + pf_->current_set;
          filter_covariance_msg weights_msg_test;
          std::vector<float> weights_test(set_test->sample_count);

          ROS_INFO("%d", set_test->sample_count);
          for(int i=0;i<set_test->sample_count;i++)
               {
                    weights_test[i] = set_test->samples[i].weight;
             if (i==0){ROS_DEBUG("Time:%f,Sample: %i, PoseX:%f, PoseY:%f, Weight:
%f",ros::Time::now().toSec(),i, set_test->samples[i].pose.v[0],set_test-
>samples[i].pose.v[1],set_test->samples[i].weight);}
          }
          weights_msg_test.cov = weights_test;
    cloud_weight_pub_beforeUpdateSensor.publish(weights_msg_test);
```

```
/*end of test pub*/

   pf_sample_set_t* set = pf_->sets + pf_->current_set;
   ROS_DEBUG("Num samples: %d\n", set->sample_count);
   //Publish number of samples
   numsamples_msg m;
   m.numsamples = set->sample_count;
   numsamples_pub_.publish(m);

/* Originally, I put the particle weight print-outs here, but all the weights get set to the same
number in the resample. Wonder if the poses matter, too??
filter_covariance_msg weights_msg;
//BEGIN SECTION for publishing cloud weight whether or not force fupdate is at play. Fingers
crossed!

std::vector<float> weights(set->sample_count);

for(int i=0;i<set->sample_count;i++)
     {
          weights[i] = set->samples[i].weight;
   ROS_DEBUG("Sample: %i, PoseX:%f, PoseY:%f, Weight: %f",i, set-
>samples[i].pose.v[0],set->samples[i].pose.v[1],set->samples[i].weight);
     }
//END SECTION for publishing cloud weight whether or not force fupdate is at play. Fingers
crossed!
weights_msg.cov = weights;
    cloud_weight_pub_.publish(weights_msg);
*/


   // Publish the resulting cloud
   // TODO: set maximum rate for publishing
   if (!m_force_update) {
     geometry_msgs::PoseArray cloud_msg;

     //filter_covariance_msg weights_msg;
     //std::vector<float> weights(set->sample_count);

     cloud_msg.header.stamp = ros::Time::now();
     cloud_msg.header.frame_id = global_frame_id_;
     cloud_msg.poses.resize(set->sample_count);
     for(int i=0;i<set->sample_count;i++)
     {
       tf::poseTFToMsg(tf::Pose(tf::createQuaternionFromYaw(set->samples[i].pose.v[2]),
                          tf::Vector3(set->samples[i].pose.v[0],
                                 set->samples[i].pose.v[1], 0)),
```

```
                cloud_msg.poses[i]);


    }
    particlecloud_pub_.publish(cloud_msg);

  }
}



if(resampled || force_publication)
{
  // Read out the current hypotheses
  double max_weight = 0.0;
  int max_weight_hyp = -1;
  std::vector<amcl_hyp_t> hyps;
  hyps.resize(pf_->sets[pf_->current_set].cluster_count);

  //publish clustercount
  cluster_count_msg c;
  c.cluster_count =pf_->sets[pf_->current_set].cluster_count;
  cluster_count_pub_.publish(c);

  for(int hyp_count = 0;
      hyp_count < pf_->sets[pf_->current_set].cluster_count; hyp_count++)
  {
    double weight;
    pf_vector_t pose_mean;
    pf_matrix_t pose_cov;
    if (!pf_get_cluster_stats(pf_, hyp_count, &weight, &pose_mean, &pose_cov))
    {
      ROS_ERROR("Couldn't get stats on cluster %d", hyp_count);
      break;
    }



    hyps[hyp_count].weight = weight;
    hyps[hyp_count].pf_pose_mean = pose_mean;
    hyps[hyp_count].pf_pose_cov = pose_cov;

    if(hyps[hyp_count].weight > max_weight)
    {
      max_weight = hyps[hyp_count].weight;
```

```
      max_weight_hyp = hyp_count;
    }



  }



  //publish max_weight of max_weigh_hyp
  //publish max_weigh_hyp
  max_weight_hyp_msg mwh;
  mwh.max_weight = max_weight;
  mwh.max_weight_hyp = max_weight_hyp;
  max_weight_hyp_pub_.publish(mwh);




  if(max_weight > 0.0)
  {
    ROS_DEBUG("Max weight pose: %.3f %.3f %.3f",
              hyps[max_weight_hyp].pf_pose_mean.v[0],
              hyps[max_weight_hyp].pf_pose_mean.v[1],
              hyps[max_weight_hyp].pf_pose_mean.v[2]);

    /*
      puts("");
      pf_matrix_fprintf(hyps[max_weight_hyp].pf_pose_cov, stdout, "%6.3f");
      puts("");
     */

    geometry_msgs::PoseWithCovarianceStamped p;
    // Fill in the header
    p.header.frame_id = global_frame_id_;
    p.header.stamp = laser_scan->header.stamp;
    // Copy in the pose
    p.pose.pose.position.x = hyps[max_weight_hyp].pf_pose_mean.v[0];
    p.pose.pose.position.y = hyps[max_weight_hyp].pf_pose_mean.v[1];

tf::quaternionTFToMsg(tf::createQuaternionFromYaw(hyps[max_weight_hyp].pf_pose_mean.v[
2]),
                          p.pose.pose.orientation);
    // Copy in the covariance, converting from 3-D to 6-D
    pf_sample_set_t* set = pf_->sets + pf_->current_set;
    for(int i=0; i<2; i++)
```

```
    {
      for(int j=0; j<2; j++)
      {
        // Report the overall filter covariance, rather than the
        // covariance for the highest-weight cluster
        //p.covariance[6*i+j] = hyps[max_weight_hyp].pf_pose_cov.m[i][j];
        p.pose.covariance[6*i+j] = set->cov.m[i][j];
      }
    }
    // Report the overall filter covariance, rather than the
    // covariance for the highest-weight cluster
    //p.covariance[6*5+5] = hyps[max_weight_hyp].pf_pose_cov.m[2][2];
    p.pose.covariance[6*5+5] = set->cov.m[2][2];

    //Publish the covariance of the filter
    //it's a 6x6 matrix
    filter_covariance_msg fcm;
//    std::vector<float> filter_covariance_vector(6);

//        for(int i=0; i<6; i++)
//        {
//        for(int j=0; j<6; j++){
//         fcm.cov[6*i+j] = set->cov.m[i][j];
//           }
//          }
//    filter_covariance_pub_.publish(fcm);

    /*
      printf("cov:\n");
      for(int i=0; i<6; i++)
      {
      for(int j=0; j<6; j++)
      printf("%6.3f ", p.covariance[6*i+j]);
      puts("");
      }
     */

    pose_pub_.publish(p);
    last_published_pose = p;

    ROS_DEBUG("New pose: %6.3f %6.3f %6.3f",
            hyps[max_weight_hyp].pf_pose_mean.v[0],
            hyps[max_weight_hyp].pf_pose_mean.v[1],
            hyps[max_weight_hyp].pf_pose_mean.v[2]);

    // subtracting base to odom from map to base and send map to odom instead
```

```
    tf::Stamped<tf::Pose> odom_to_map;
    try
    {
      tf::Transform
tmp_tf(tf::createQuaternionFromYaw(hyps[max_weight_hyp].pf_pose_mean.v[2]),
                        tf::Vector3(hyps[max_weight_hyp].pf_pose_mean.v[0],
                                    hyps[max_weight_hyp].pf_pose_mean.v[1],
                                    0.0));
      tf::Stamped<tf::Pose> tmp_tf_stamped (tmp_tf.inverse(),
                                            laser_scan->header.stamp,
                                            base_frame_id_);
      this->tf_->transformPose(odom_frame_id_,
                               tmp_tf_stamped,
                               odom_to_map);
    }
    catch(tf::TransformException)
    {
      ROS_DEBUG("Failed to subtract base to odom transform");
      return;
    }

    latest_tf_ = tf::Transform(tf::Quaternion(odom_to_map.getRotation()),
                               tf::Point(odom_to_map.getOrigin()));
    latest_tf_valid_ = true;

    if (tf_broadcast_ == true)
    {
      // We want to send a transform that is good up until a
      // tolerance time so that odom can be used
      ros::Time transform_expiration = (laser_scan->header.stamp +
                                        transform_tolerance_);
      tf::StampedTransform tmp_tf_stamped(latest_tf_.inverse(),
                                          transform_expiration,
                                          global_frame_id_, odom_frame_id_);
      this->tfb_->sendTransform(tmp_tf_stamped);
      sent_first_transform_ = true;
    }
  }
  else
  {
    ROS_ERROR("No pose!");
  }
 }
 else if(latest_tf_valid_)
 {
  if (tf_broadcast_ == true)
```

```
    {
      // Nothing changed, so we'll just republish the last transform, to keep
      // everybody happy.
      ros::Time transform_expiration = (laser_scan->header.stamp +
                                        transform_tolerance_);
      tf::StampedTransform tmp_tf_stamped(latest_tf_.inverse(),
                                          transform_expiration,
                                          global_frame_id_, odom_frame_id_);
      this->tfb_->sendTransform(tmp_tf_stamped);
    }

    // Is it time to save our last pose to the param serverf
    ros::Time now = ros::Time::now();
    if((save_pose_period.toSec() > 0.0) &&
       (now - save_pose_last_time) >= save_pose_period)
    {
      this->savePoseToServer();
      save_pose_last_time = now;
    }
  }

}

double
AmclNode::getYaw(tf::Pose& t)
{
  double yaw, pitch, roll;
  t.getBasis().getEulerYPR(yaw,pitch,roll);
  return yaw;
}

void
AmclNode::initialPoseReceived(const geometry_msgs::PoseWithCovarianceStampedConstPtr&
msg)
{
  handleInitialPoseMessage(*msg);
}

void
AmclNode::handleInitialPoseMessage(const geometry_msgs::PoseWithCovarianceStamped&
msg)
{
  boost::recursive_mutex::scoped_lock prl(configuration_mutex_);
  if(msg.header.frame_id == "")
  {
    // This should be removed at some point
```

```
   ROS_WARN("Received initial pose with empty frame_id.  You should always supply a
frame_id.");
  }
  // We only accept initial pose estimates in the global frame, #5148.
  else if(tf_->resolve(msg.header.frame_id) != tf_->resolve(global_frame_id_))
  {
   ROS_WARN("Ignoring initial pose in frame \"%s\"; initial poses must be in the global frame,
\"%s\"",
           msg.header.frame_id.c_str(),
           global_frame_id_.c_str());
   return;
  }

  // In case the client sent us a pose estimate in the past, integrate the
  // intervening odometric change.
  tf::StampedTransform tx_odom;
  try
  {
   ros::Time now = ros::Time::now();
   // wait a little for the latest tf to become available
   tf_->waitForTransform(base_frame_id_, msg.header.stamp,
                         base_frame_id_, now,
                         odom_frame_id_, ros::Duration(0.5));
   tf_->lookupTransform(base_frame_id_, msg.header.stamp,
                         base_frame_id_, now,
                         odom_frame_id_, tx_odom);
  }
  catch(tf::TransformException e)
  {
   // If we've never sent a transform, then this is normal, because the
   // global_frame_id_ frame doesn't exist.  We only care about in-time
   // transformation for on-the-move pose-setting, so ignoring this
   // startup condition doesn't really cost us anything.
   if(sent_first_transform_)
   tx_odom.setIdentity();
  }

  tf::Pose pose_old, pose_new;
  tf::poseMsgToTF(msg.pose.pose, pose_old);
  pose_new = pose_old * tx_odom;

  // Transform into the global frame

  ROS_INFO("Setting pose (%.6f): %.3f %.3f %.3f",
         ros::Time::now().toSec(),
         pose_new.getOrigin().x(),
```

```
          pose_new.getOrigin().y(),
          getYaw(pose_new));
  // Re-initialize the filter
  pf_vector_t pf_init_pose_mean = pf_vector_zero();
  pf_init_pose_mean.v[0] = pose_new.getOrigin().x();
  pf_init_pose_mean.v[1] = pose_new.getOrigin().y();
  pf_init_pose_mean.v[2] = getYaw(pose_new);
  pf_matrix_t pf_init_pose_cov = pf_matrix_zero();
  // Copy in the covariance, converting from 6-D to 3-D
  for(int i=0; i<2; i++)
  {
    for(int j=0; j<2; j++)
    {
      pf_init_pose_cov.m[i][j] = msg.pose.covariance[6*i+j];
    }
  }
  pf_init_pose_cov.m[2][2] = msg.pose.covariance[6*5+5];

  delete initial_pose_hyp_;
  initial_pose_hyp_ = new amcl_hyp_t();
  initial_pose_hyp_->pf_pose_mean = pf_init_pose_mean;
  initial_pose_hyp_->pf_pose_cov = pf_init_pose_cov;
  applyInitialPose();
}

/**
 * If initial_pose_hyp_ and map_ are both non-null, apply the initial
 * pose to the particle filter state.  initial_pose_hyp_ is deleted
 * and set to NULL after it is used.
 */
void
AmclNode::applyInitialPose()
{
  boost::recursive_mutex::scoped_lock cfl(configuration_mutex_);
  if( initial_pose_hyp_ != NULL && map_ != NULL ) {
    pf_init(pf_, initial_pose_hyp_->pf_pose_mean, initial_pose_hyp_->pf_pose_cov);
    pf_init_ = false;

    delete initial_pose_hyp_;
    initial_pose_hyp_ = NULL;
  }
}
```

Gazebo C++ Plugin Created to Inject Robot Kidnapping or Fault Events into Simulation

```
#include <ros/ros.h>
//#include "/home/ebrenna8/gazebo_ros_api_plugin.h"
#include
"/home/ebrenna8/gazebo_downloads/src/gazebo_ros_pkgs/gazebo_ros/include/gazebo_ros/gazeb
o_ros_api_plugin.h"
//#include "/home/ebrenna8/usr/include/gazebo-2.2/gazebo/physics/Physics.hh"
//will need to add the gazebo header here, I think
//#include "gazebo_ros/gazebo_ros_api_plugin.h"
#include <gazebo_msgs/SpawnModel.h>
#include <iostream>
#include <fstream>
//#include "/usr/include/gazebo-2.2/gazebo/transport/Node.hh"
//#include "gazebo/transport/TransportIface.hh"
//#include "gazebo/transport/Publisher.hh"
//#include "gazebo/transport/Subscriber.hh"
#include "gazebo/physics/physics.hh"
#include <gazebo/gazebo.hh>
#include "gazebo/msgs/MessageTypes.hh"
#include "std_msgs/String.h"
#include <exception>
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
#include <gazebo/msgs/msgs.hh>
#include <tf2_ros/transform_listener.h>
#include <geometry_msgs/TransformStamped.h>
#include <geometry_msgs/Twist.h>
#include <gazebo_msgs/SetModelState.h>
#include "/opt/ros/indigo/include/tf2_ros/buffer.h"

#include "geometry_msgs/PointStamped.h"
#include "tf2_ros/transform_listener.h"
#include "tf2_ros/message_filter.h"
#include "message_filters/subscriber.h"
#include "tf2_geometry_msgs/tf2_geometry_msgs.h"
#include <math.h>

#include <tf/transform_listener.h>
#include <geometry_msgs/Twist.h>
#include <tf/transform_datatypes.h>
#include "ros/callback_queue.h"

#include <gazebo_plugins/gazebo_ros_utils.h>
#include <thread>
#include <cmath>
```

```
#include <limits>
#include <vector>
//include <tf2_geometry_msgs/tf2_geometry_msgs.h>
//#include "tf2_geometry_msgs/tf2_geometry_msgs.h"
//#include "/usr/include/gazebo-2.2/gazebo/common/common.hh"
//#include "/usr/include/gazebo-2.2/gazebo/gazebo.hh"
//#include "/usr/include/gazebo-2.2/gazebo/transport/transport.hh"

namespace gazebo
{



//typedef const boost::shared_ptr<const tf2_msgs::TFMessage> TFMessagePtr;

class CallSpawnDeleteWorldPlugin:public WorldPlugin
{
        //public: CallSpawnDeleteWorldPlugin():WorldPlugin()
//{
//        printf("Hello World!\n");


//        printf("before anything happens:...");
        /*transport::run();
        transport::NodePtr node(new gazebo::transport::Node()); //transport::Node()
        //node->Init("/home/ebrenna8/turtlebot_gazebo/mdd.world");
        printf("Here!");
        node->Init();
        printf("ABC");
        //i bet mobile_base doesn't exist yet.
        //transport::requestNoReply(node, "entity_delete", "mobile_base");*/


        /*msgs::Request *msg = msgs::CreateRequest("entity_delete", "mobile_base");
        printf("Here");
        transport::PublisherPtr pub = node->Advertise<msgs::Request>("~/request");
        pub->WaitForConnection();
        printf("wait for connection");
        pub->Publish(*msg);
        printf("publish");
        delete msg;
        printf("deletin msg");
        transport::fini();*/
//        printf("end");
//}//end constructor*/
```

```
//I think you can make the world/sdf accessible globally thru Load
private: physics::WorldPtr world;
private: sdf::ElementPtr sdf;
private:ros::NodeHandle *rosNode;
private:tf2_ros::Buffer tfBuffer; //tf2_ros
//private:tf::TransformListener listener;

private:tf::StampedTransform transformLeft;
private:tf::StampedTransform transformRight;
private:tf::StampedTransform transform;

//rosrun tf tf_echo "wheel_right_link" "odom"
//init position when started at (0,0)
private:float prev_x_right = -0.0;
private:float prev_y_right = 0.035;
private:float prev_theta_right = 0.005;


//rosrun tf tf_echo "wheel_left_link" "odom"
//init position when started at (0,0)
private:float prev_x_left = 0.0;
private:float prev_y_left = 0.035;
private:float prev_theta_left = -0.003;

private:float angular_vel_right = 0.0;
private:float angular_vel_left = 0.0;
private:float linear_vel_right = 0.0;
private:float linear_vel_left = 0.0;
private:float linear_vel_right_flat_tire = 0.0;

private:float wheel_radius_left = 0.0;
private:float wheel_radius_right = 0.0;
private:double theta = 0.0;

private:float x_star_left = 0.0;
private:float y_star_left = 0.0;
private:float x_star_right = 0.0;
private:float y_star_right = 0.0;
private:tf2_msgs::TFMessage t;

private:float current_x_right = 0.0;
private:float current_y_right = 0.0;
private:double current_theta_right = 0.0;

private:float current_x_left = 0.0;
private:float current_y_left = 0.0;
```

```
private:double current_theta_left = 0.0;

private:float x_prime = 0.0;
private:float y_prime = 0.0;
private:float theta_prime = 0.0;

private:float x_prime_flat_tire = 0.0;
private:float y_prime_flat_tire = 0.0;
private:float theta_prime_flat_tire = 0.0;

private:float current_x = 0.0;
private:float current_y = 0.0;
private:float current_z = 0.0;
private:double current_theta = 0.0;
private:double current_tf_theta = 0.0;

private:double prev_x = 0.0;
private:double prev_y = 0.0;
private:double prev_theta = 0.0;

private:double prev_x_2 = 0.0;
private:double prev_y_2 = 0.0;
private:double prev_theta_2 = 0.0;

private:double x_motion_update = 0.0;
private:double y_motion_update = 0.0;
private:double theta_motion_update = 0.0;


private:double prev_x_motion_update = 0.0;
private:double prev_y_motion_update = 0.0;
private:double prev_theta_motion_update = 0.0;

private:ros::Publisher kidnap_pub;
private:std_msgs::String kidnap_msg;

//private:tf2_ros::TransformListener tfListener(tfBuffer);

private: ros::CallbackQueue callbackQueue;

private: GazeboRosPtr gazeboRos;
private: ros::Subscriber joySub;

private: std::unique_ptr<ros::NodeHandle> rosNodeTF;
private: std::thread rosQueueThread;
private:ros::ServiceClient client;
```

```
private:ros::ServiceClient Getclient;
private:ros::ServiceClient GlobalLocalizationClient;

public: CallSpawnDeleteWorldPlugin() {} //brief constructor

private:double count;

private: std::ofstream myGazeboFile;
private: std::ofstream myTFFile;
private:ros::Timer timer;

//void timerCallback(const ros::TimerEvent& event);

public: virtual void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf)
{

        //transport::NodePtr node;
        this->world = _world;
        this->sdf = _sdf;

        this->node = transport::NodePtr(new transport::Node());
        this->node->Init(_world->GetName());
//      printf("haven't died yet");
//      std::cout<<_world->GetName();
        //Good through here.


        std::string topicname = "/tf";


        this->rosNode = new ros::NodeHandle(_world->GetName());

        //ros::SubscribeOptions subOpts =
ros::SubscribeOptions::create<tf2_msgs::TFMessage>("/tf", 1,
boost::bind(&CallSpawnDeleteWorldPlugin::OnMsgTest, this, _1), ros::VoidPtr(), &this-
>callbackQueue);


        kidnap_pub = rosNode->advertise<std_msgs::String>("kidnap_event",1000);
        //Creating a timer.


        //this->timer = this->rosNode->createTimer(ros::Duration(1),
boost::bind(&CallSpawnDeleteWorldPlugin::timerCallback, this, _1));//this creates a continuous
localization disturbance
```

```
        //this->timer = this->rosNode->createTimer(ros::Duration(45),
boost::bind(&CallSpawnDeleteWorldPlugin::timerCallbackMinor, this, _1));//for orientation

        //this->timer = this->rosNode->createTimer(ros::Duration(1),
boost::bind(&CallSpawnDeleteWorldPlugin::timerCallbackRelocationUnitTest, this, _1));


        this->timer = this->rosNode->createTimer(ros::Duration(35),
boost::bind(&CallSpawnDeleteWorldPlugin::timerCallbackMajor, this, _1)); //45 (Major 1-10?
11?) and 65.

//      this->timer = this->rosNode->createTimer(ros::Duration(35),
boost::bind(&CallSpawnDeleteWorldPlugin::timerCallbackMajorWithGlobalLocalization, this,
_1)); //45 (Major 1-10? 11?) and 65.

        //Explanation: You have to do the Boost::bind ...s stuff to pass the function in the right
form.


        //Subscribe to TF
//      this->joySub = this->rosNode->subscribe(subOpts);
        this-
>rosQueueThread=std::thread(std::bind(&CallSpawnDeleteWorldPlugin::QueueThread, this));


        this->client = this->rosNode-
>serviceClient<gazebo_msgs::SetModelState>("/gazebo/set_model_state");

        this->Getclient = this->rosNode-
>serviceClient<gazebo_msgs::GetModelState>("/gazebo/get_model_state");

        this->GlobalLocalizationClient = this->rosNode-
>serviceClient<std_srvs::Empty>("/global_localization");

        this->count = 0;
        this->myGazeboFile.open("/home/ebrenna8/Gazebo.txt");
        this->myTFFile.open("/home/ebrenna8/TF.txt");
        this->myGazeboFile
<<"RosTime\tprev_x_2\tprev_y_2\tprev_theta_2\tprev_x\tprev_y\tprev_theta\tcurrent_x\tcurrent
_y\tcurrent_theta\tNewX\tNewY\tNewTheta";



        //transport::SubscriberPtr commandSubscriber;
```

```
	//commandSubscriber = node->Subscribe("~/my_topic",&CallSpawnDeleteWorldPlugin::callbackFunction, this);

//********BELOW IS FROM TRYING TO USE ROS MESSAGES.******
	//std::unique_ptr<ros::NodeHandle> rosNode;
	//ros::NodeHandle rosNode;
	//ros::Subscriber rosSub;
	//ros::CallbackQueue rosQueue;
	//std::thread rosQueueThread;
	//boost::thread rosQueueThread; //maybe include <thread>

	/*if (!ros::isInitialized())
	{
		int argc =0;
		char **argv = NULL;
		ros::init(argc, argv, "gazebo_client", ros::init_options::NoSigintHandler);
	}

	rosNode.reset(new ros::NodeHandle("gazebo_client"));
	ros::SubscribeOptions so =
		ros::SubscribeOptions::create<std_msgs::String>("/"+_world->GetName() + "/vel_cmd",
		1,
		boost::bind(&CallSpawnDeleteWorldPlugin::OnRosMsg, this, _1),
		ros::VoidPtr(), &rosQueue);
	this->rosQueueThread = std::thread(std::bind(&CallSpawnDeleteWorldPlugin::QueueThread, this));
*/
//*******END OF SECTION FOR ROS MESSAGES***********


//	printf("Subscribed to my_topic");
	//maybe this goes into the callback
	/*msgs::Request *msg = msgs::CreateRequest("entity_delete", "mobile_base");
	printf("Here");
	transport::PublisherPtr pub = node->Advertise<msgs::Request>("~/request");
	pub->WaitForConnection();
	printf("wait for connection");
	pub->Publish(*msg);
	printf("publish");
	delete msg;
	printf("deletin msg");
	transport::fini();*/


/*FUCNTIONS THAT WENT WITH USING ROS MESSAGES
```

```
        public:void OnRosMsg(const std_msgs::String &_msg)
        {
                printf("Message received!!!");
        }

        private: void QueueThread() {
                static const double timeout = 0.01;
                while (this->rosNode->ok())
                {
                        this->rosQueue.callAvailable(ros::WallDuration(timeout));
                }
        }*/


}//end load

//private: void OnMsg(ConstVector3dPtr &msg)

//private: void OnMsg(const tf2_msgs::TFMessage::ConstPtr& &msg)

private: void QueueThread()
{
        static const double timeout = 0.10;
        while (this->rosNode->ok())
        {
                this ->callbackQueue.callAvailable(ros::WallDuration(timeout));
        }
}

private:double roll, pitch, yaw;
private:geometry_msgs::Pose newPose;
private:std::string s;
private:bool found_base_footprint = false;
private:bool found_base_link = false;
private:bool found_wheel_right_link = false;
private:bool found_wheel_left_link = false;

private:tf::Transformer TFtransformer;
/*private:int vel_direction = 0;
private:int vel_direction_left = 0;
private:int vel_direction_right = 0;
*/

private:double vel_direction = 0;
private:double vel_direction_left = 0;
private:double vel_direction_right = 0;
```

```
private:gazebo::common::Time current_time;
private:gazebo::common::Time prev_time;
private:double d;
private:double prev_predicted_theta = 0.0;
private:double prev_predicted_x = 0.0;
private:double prev_predicted_y = 0.0;

private:double current_tf_x = 0.0;
private:double current_tf_y = 0.0;

private:double slip_x_motion_update = 0.0;
private:double slip_y_motion_update = 0.0;
private:double slip_theta_motion_update = 0.0;

private:double slip_x_prime = 0.0;
private:double slip_y_prime = 0.0;
private:double slip_theta_prime = 0.0;

private:double prev_slip_x_motion_update = 0.0;
private:double prev_slip_y_motion_update = 0.0;
private:double prev_slip_theta_motion_update = 0.0;
private:double *intx_inty =NULL;
private:double *xc_yc_mainradius = NULL;
private:double *newX_newY_newTheta=NULL;
private:ros::Time callback_time;
private:int icount=0;


//Function for Timer-controller callback:
void timerCallback(const ros::TimerEvent& event){
//private:void timerCallback(void){

//private:void timerCallback(const ros::TimerEvent& event);
//ROS_INFO("INSIDE TIMER CALLBACK!");
callback_time = ros::Time::now();

        ros::Time five_seconds(5.0);


        ros::Time lower_seconds(40.0);
        ros::Time upper_seconds(50.0);


        if (ros::Time::now() > five_seconds) { //This is to prevetn the callback from inhibiting
the inital load of Gazebo.
```

```
		if (ros::Time::now() >= lower_seconds && ros::Time::now() <= upper_seconds){//Begin execute only if meets time

		this->world->SetPaused(true);



		//Gazebo model pose:
		gazebo_msgs::GetModelState getmodelstate;
		getmodelstate.request.model_name = "mobile_base";
		this->Getclient.call(getmodelstate);

		current_x = getmodelstate.response.pose.position.x;
		current_y = getmodelstate.response.pose.position.y;
		current_z = getmodelstate.response.pose.position.z;

		tf::Quaternion q(getmodelstate.response.pose.orientation.x, getmodelstate.response.pose.orientation.y, getmodelstate.response.pose.orientation.z, getmodelstate.response.pose.orientation.w);
		tf::Matrix3x3 m(q);
		m.getRPY(roll, pitch, yaw);
		current_theta = yaw;


		//If it's the first time, set prev_x to current pos
		if (prev_x == 0){
			prev_x = current_x;
			prev_y = current_y;
			prev_theta = current_theta;

			prev_x_2 = prev_x;
			prev_y_2 = prev_y;
			prev_theta_2 = prev_theta;
		}
		//NEW ARC-BASED APPROACH!!!
		//RESETTING X_PRIME, Y_PRIME, AND THETA_PRIME

		ROS_INFO("Inputting:");
		ROS_INFO("{%f, %f, %f, %f, %f, %f, %f, %f, %f};", prev_x_2, prev_y_2, prev_theta_2, prev_x, prev_y, prev_theta, current_x, current_y, current_theta);

		//0.0002 seems to be a good metric for moving vs not moving
		if ((std::abs(current_x - prev_x) > 0.0002) && (std::abs(current_y-prev_y) > 0.0002)){ //if there's been sufficient movement
```

```
                icount = icount + 1;

                this->myGazeboFile<<"\nKidnapped Pose!";

                this->myGazeboFile<<"\n"<<callback_time<<"\t"<<prev_x_2<<"\t"<<prev_y_2<<"\t"<<prev_theta_2<<"\t"<<prev_x<<"\t"<<prev_y<<"\t"<<prev_theta<<"\t"<<current_x<<"\t"<<current_y<<"\t"<<current_theta;


                std::vector<double> r(3);
                r = calculate_new_pose(prev_theta_2, prev_x_2, prev_y_2, prev_theta, prev_x, prev_y, current_theta, current_x, current_y);


                this->myGazeboFile<<"\t"<<r[0]<<"\t"<<r[1]<<"\t"<<r[2];

                //this->myTFFile<<"\t"<<r[0]<<"\t"<<r[1]<<"\t"<<r[2];

                ROS_INFO("Calculated pose:");
                ROS_INFO("%f %f %f", r[0], r[1], r[2]);
                x_prime = r[0];
                y_prime = r[1];
                theta_prime = r[2];


                //END OF ARC-BASED APPROACH!!!

                //Prepare to publish

                newPose.position.x =x_prime;//prev_predicted_x; //_flat_tire;
                newPose.position.y =y_prime;//prev_predicted_y; //_flat_tire;



                //Unroll theta_prime into a quaternion
                tf::Quaternion quat;
                quat.setRPY(0, 0, theta_prime);//theta_prime);//prev_predicted_theta);//ROLL, PITCH *YAW*

                newPose.orientation.x = quat.x();
                newPose.orientation.y = quat.y();
                newPose.orientation.z = quat.z();
                newPose.orientation.w = quat.w();
```

```
                        //call gazebo set model state:

                        gazebo_msgs::ModelState modelstate;
                        modelstate.model_name = (std::string) "mobile_base";
                        modelstate.reference_frame = (std::string) "world";
                        modelstate.pose = newPose;

                        gazebo_msgs::SetModelState setmodelstate;
                        setmodelstate.request.model_state = modelstate;


                        if (std::isnan(modelstate.pose.position.x) ||
std::isnan(modelstate.pose.position.y) || std::isnan(modelstate.pose.position.z) ||
std::isnan(modelstate.pose.orientation.x) ||std::isnan(modelstate.pose.orientation.y) ||
std::isnan(modelstate.pose.orientation.z) || std::isnan(modelstate.pose.orientation.w)
||std::isnan(modelstate.twist.linear.x) || std::isnan(modelstate.twist.linear.y) ||
std::isnan(modelstate.twist.linear.z) ||std::isnan(modelstate.twist.angular.x) ||
std::isnan(modelstate.twist.angular.y) ||std::isnan(modelstate.twist.angular.z)){
                                ROS_INFO("There's a nan here...1");
                                this->world->SetPaused(false);

                                prev_x_2 = prev_x;
                                prev_y_2 = prev_y;
                                prev_theta_2 = prev_theta;
                                prev_x = current_x;
                                prev_y = current_y;
                                prev_theta = current_theta;
                                }

                        else{
                                if(std::isnan(r[0]) || std::isnan(r[1]) || std::isnan(r[2])) {
                                        ROS_INFO("********** Missed a NAN
**********\r\n");
                                }
                                //ROS_INFO("%f", ros::Time::now().toSec());
                                this->client.call(setmodelstate);//coment out if you DO NOT want
to PUBLISH
                                //ROS_INFO("%f", ros::Time::now().toSec());

                                prev_x_2 = prev_x;
                                prev_y_2 = prev_y;
                                prev_theta_2 = prev_theta;

                                prev_x = x_prime;//current_x;
                                prev_y = y_prime;//current_y;
```

```
                        prev_theta = theta_prime;//current_theta;

                }



        }//end of if there was sufficient movement
        else{

                this->myGazeboFile<<"\n"<<callback_time<<"\t"<<prev_x_2<<"\t"<<prev_y_2<<"\t"<<prev_theta_2<<"\t"<<prev_x<<"\t"<<prev_y<<"\t"<<prev_theta<<"\t"<<current_x<<"\t"<<current_y<<"\t"<<current_theta<<"\tInsufficient Movement\tInsufficient Movement\tInsufficientMovement";

                prev_x_2 = prev_x;
                prev_y_2 = prev_y;
                prev_theta_2 = prev_theta;


                prev_x = current_x;
                prev_y = current_y;
                prev_theta = current_theta;
        }



        this->world->SetPaused(false);

        }//End of time constraints for what happens

    }//end of 5 seconds thing
}//end of timerCallback



void timerCallbackMinor(const ros::TimerEvent& event){

    callback_time = ros::Time::now();

    this->world->SetPaused(true);

    this->timer.stop();//so that it only fires once
```

```
        //Gazebo model pose:
        gazebo_msgs::GetModelState getmodelstate;
        getmodelstate.request.model_name = "mobile_base";
        this->Getclient.call(getmodelstate);

        current_x = getmodelstate.response.pose.position.x;
        current_y = getmodelstate.response.pose.position.y;
        current_z = getmodelstate.response.pose.position.z;

        tf::Quaternion q(getmodelstate.response.pose.orientation.x,
getmodelstate.response.pose.orientation.y, getmodelstate.response.pose.orientation.z,
getmodelstate.response.pose.orientation.w);
        tf::Matrix3x3 m(q);
        m.getRPY(roll, pitch, yaw);
        current_theta = yaw;




        x_prime = current_x;// + 0.1;
        y_prime = current_y;// + 0.1;
        ROS_INFO("Minor Kidnapping. Moved the robot from %f, %f to %f, %f", current_x,
current_y, x_prime, y_prime);
        //theta_prime = current_theta;// + 4.08;//10deg = 0.17 rad//original value
        //theta_prime = current_theta + 2.08; //120 degrees
        theta_prime = current_theta + 3.3; //190 degrees




        newPose.position.x =x_prime;//prev_predicted_x; //_flat_tire;
        newPose.position.y =y_prime;//prev_predicted_y; //_flat_tire;



        //Unroll theta_prime into a quaternion
        tf::Quaternion quat;
        quat.setRPY(0, 0, theta_prime);//theta_prime);//prev_predicted_theta);//ROLL, PITCH
*YAW*

        newPose.orientation.x = quat.x();
        newPose.orientation.y = quat.y();
        newPose.orientation.z = quat.z();
```

```
newPose.orientation.w = quat.w();



//call gazebo set model state:
gazebo_msgs::ModelState modelstate;
modelstate.model_name = (std::string) "mobile_base";
modelstate.reference_frame = (std::string) "world";
modelstate.pose = newPose;
gazebo_msgs::SetModelState setmodelstate;
setmodelstate.request.model_state = modelstate;


if (std::isnan(modelstate.pose.position.x) || std::isnan(modelstate.pose.position.y) ||
std::isnan(modelstate.pose.position.z) || std::isnan(modelstate.pose.orientation.x)
||std::isnan(modelstate.pose.orientation.y) || std::isnan(modelstate.pose.orientation.z) ||
std::isnan(modelstate.pose.orientation.w) ||std::isnan(modelstate.twist.linear.x) ||
std::isnan(modelstate.twist.linear.y) || std::isnan(modelstate.twist.linear.z)
||std::isnan(modelstate.twist.angular.x) || std::isnan(modelstate.twist.angular.y)
||std::isnan(modelstate.twist.angular.z)){
        ROS_INFO("There's a nan here...1");
        this->world->SetPaused(false);

        prev_x_2 = prev_x;
        prev_y_2 = prev_y;
        prev_theta_2 = prev_theta;
        prev_x = current_x;
        prev_y = current_y;
        prev_theta = current_theta;
        }

else{
        //ROS_INFO("%f", ros::Time::now().toSec());
        this->client.call(setmodelstate);//coment out if you DO NOT want to PUBLISH
        ROS_INFO("%f", ros::Time::now().toSec());
        ROS_INFO("CHANGED THETA FROM %f to %f", current_theta, theta_prime);
        this->myGazeboFile<<callback_time<<std::endl;
        prev_x_2 = prev_x;
        prev_y_2 = prev_y;
        prev_theta_2 = prev_theta;
        prev_x = x_prime;//current_x;
        prev_y = y_prime;//current_y;
        prev_theta = theta_prime;//current_theta;

        //Send a message that the event occurred.
        kidnap_msg.data = "KIDNAP OCCURRED.";
```

```
                kidnap_pub.publish(kidnap_msg);

                }




        this->world->SetPaused(false);

//      }//End of time constraints for what happens


}//end of timerCallbackMinor


void timerCallbackMajorOffMap(const ros::TimerEvent& event){
        ROS_INFO("INSIDE MAJOR EVENT TIMER CALLBACK!");

        callback_time = ros::Time::now();

        this->world->SetPaused(true);

        this->timer.stop();//so that it only fires once


        //Gazebo model pose:
        gazebo_msgs::GetModelState getmodelstate;
        getmodelstate.request.model_name = "mobile_base";
        this->Getclient.call(getmodelstate);

        current_x = getmodelstate.response.pose.position.x;
        current_y = getmodelstate.response.pose.position.y;
        current_z = getmodelstate.response.pose.position.z;

        tf::Quaternion q(getmodelstate.response.pose.orientation.x,
getmodelstate.response.pose.orientation.y, getmodelstate.response.pose.orientation.z,
getmodelstate.response.pose.orientation.w);
        tf::Matrix3x3 m(q);
        m.getRPY(roll, pitch, yaw);
        current_theta = yaw;
```

```
        x_prime = current_x +1000;//+ 0.2;
        y_prime = current_y+1000;//+0.2;
        theta_prime = current_theta + 0.5;//10deg = 0.17 rad

        //x_prime = 0;
        //y_prime = 0;
        //theta_prime = 3.14;



        newPose.position.x =x_prime;//prev_predicted_x; //_flat_tire;
        newPose.position.y =y_prime;//prev_predicted_y; //_flat_tire;



        //Unroll theta_prime into a quaternion
        tf::Quaternion quat;
        quat.setRPY(0, 0, theta_prime);//theta_prime);//prev_predicted_theta);//ROLL, PITCH
*YAW*

        newPose.orientation.x = quat.x();
        newPose.orientation.y = quat.y();
        newPose.orientation.z = quat.z();
        newPose.orientation.w = quat.w();



        //call gazebo set model state:
        gazebo_msgs::ModelState modelstate;
        modelstate.model_name = (std::string) "mobile_base";
        modelstate.reference_frame = (std::string) "world";
        modelstate.pose = newPose;
        gazebo_msgs::SetModelState setmodelstate;
        setmodelstate.request.model_state = modelstate;


        if (std::isnan(modelstate.pose.position.x) || std::isnan(modelstate.pose.position.y) ||
std::isnan(modelstate.pose.position.z) || std::isnan(modelstate.pose.orientation.x)
||std::isnan(modelstate.pose.orientation.y) || std::isnan(modelstate.pose.orientation.z) ||
std::isnan(modelstate.pose.orientation.w) ||std::isnan(modelstate.twist.linear.x) ||
std::isnan(modelstate.twist.linear.y) || std::isnan(modelstate.twist.linear.z)
||std::isnan(modelstate.twist.angular.x) || std::isnan(modelstate.twist.angular.y)
||std::isnan(modelstate.twist.angular.z)){
                ROS_INFO("There's a nan here...1");
```

```
                this->world->SetPaused(false);

                prev_x_2 = prev_x;
                prev_y_2 = prev_y;
                prev_theta_2 = prev_theta;
                prev_x = current_x;
                prev_y = current_y;
                prev_theta = current_theta;
                }

        else{

                //ROS_INFO("%f", ros::Time::now().toSec());
                this->client.call(setmodelstate);//coment out if you DO NOT want to PUBLISH
                //ROS_INFO("%f", ros::Time::now().toSec());
                ROS_INFO("Changed X and y from %f , %f to %f , %f",current_x, current_y,
x_prime, y_prime);
                ROS_INFO("CHANGED THETA FROM %f to %f", current_theta, theta_prime);
                this->myGazeboFile<<callback_time<<std::endl;
                prev_x_2 = prev_x;
                prev_y_2 = prev_y;
                prev_theta_2 = prev_theta;
                prev_x = x_prime;//current_x;
                prev_y = y_prime;//current_y;
                prev_theta = theta_prime;//current_theta;

//Send a message that the event occurred.
                kidnap_msg.data = "KIDNAP OCCURRED.";
                ROS_INFO("SENDING MESSAGE");
                kidnap_pub.publish(kidnap_msg);

                }







        this->world->SetPaused(false);

        //}//End of time constraints for what happens


}//end of timerCallbackMajorOffMap
```

```
void timerCallbackMajorWithGlobalLocalization(const ros::TimerEvent& event){
	ROS_INFO("INSIDE MAJOR EVENT TIMER CALLBACK!");

	callback_time = ros::Time::now();

	this->world->SetPaused(true);

	this->timer.stop();//so that it only fires once


	//Gazebo model pose:
	gazebo_msgs::GetModelState getmodelstate;
	getmodelstate.request.model_name = "mobile_base";
	this->Getclient.call(getmodelstate);

	current_x = getmodelstate.response.pose.position.x;
	current_y = getmodelstate.response.pose.position.y;
	current_z = getmodelstate.response.pose.position.z;

	tf::Quaternion q(getmodelstate.response.pose.orientation.x,
getmodelstate.response.pose.orientation.y, getmodelstate.response.pose.orientation.z,
getmodelstate.response.pose.orientation.w);
	tf::Matrix3x3 m(q);
	m.getRPY(roll, pitch, yaw);
	current_theta = yaw;




	x_prime = -1;//current_x +1000;//+ 0.2;
	y_prime =-2;//current_y+1000;//+0.2;
	theta_prime = current_theta;// + 0.5;//10deg = 0.17 rad

	//x_prime = 0;
	//y_prime = 0;
	//theta_prime = 3.14;



	newPose.position.x =x_prime;//prev_predicted_x; //_flat_tire;
	newPose.position.y =y_prime;//prev_predicted_y; //_flat_tire;
```

```
        //Unroll theta_prime into a quaternion
        tf::Quaternion quat;
        quat.setRPY(0, 0, theta_prime);//theta_prime);//prev_predicted_theta);//ROLL, PITCH
*YAW*

        newPose.orientation.x = quat.x();
        newPose.orientation.y = quat.y();
        newPose.orientation.z = quat.z();
        newPose.orientation.w = quat.w();



        //call gazebo set model state:
        gazebo_msgs::ModelState modelstate;
        modelstate.model_name = (std::string) "mobile_base";
        modelstate.reference_frame = (std::string) "world";
        modelstate.pose = newPose;
        gazebo_msgs::SetModelState setmodelstate;
        setmodelstate.request.model_state = modelstate;


        if (std::isnan(modelstate.pose.position.x) || std::isnan(modelstate.pose.position.y) ||
std::isnan(modelstate.pose.position.z) || std::isnan(modelstate.pose.orientation.x)
||std::isnan(modelstate.pose.orientation.y) || std::isnan(modelstate.pose.orientation.z) ||
std::isnan(modelstate.pose.orientation.w) ||std::isnan(modelstate.twist.linear.x) ||
std::isnan(modelstate.twist.linear.y) || std::isnan(modelstate.twist.linear.z)
||std::isnan(modelstate.twist.angular.x) || std::isnan(modelstate.twist.angular.y)
||std::isnan(modelstate.twist.angular.z)){
                ROS_INFO("There's a nan here...1");
                this->world->SetPaused(false);

                prev_x_2 = prev_x;
                prev_y_2 = prev_y;
                prev_theta_2 = prev_theta;
                prev_x = current_x;
                prev_y = current_y;
                prev_theta = current_theta;
                }

        else{

                //ROS_INFO("%f", ros::Time::now().toSec());
                this->client.call(setmodelstate);//coment out if you DO NOT want to PUBLISH
```

```
                std_srvs::Empty emptyArg;
                ROS_INFO("Calling global localizaiton");
                this->GlobalLocalizationClient.call(emptyArg);


                //ROS_INFO("%f", ros::Time::now().toSec());
                ROS_INFO("Changed X and y from %f , %f to %f , %f",current_x, current_y,
x_prime, y_prime);
                ROS_INFO("CHANGED THETA FROM %f to %f", current_theta, theta_prime);
                this->myGazeboFile<<callback_time<<std::endl;
                prev_x_2 = prev_x;
                prev_y_2 = prev_y;
                prev_theta_2 = prev_theta;
                prev_x = x_prime;//current_x;
                prev_y = y_prime;//current_y;
                prev_theta = theta_prime;//current_theta;

//Send a message that the event occurred.
                kidnap_msg.data = "KIDNAP OCCURRED.";
                ROS_INFO("SENDING MESSAGE");
                kidnap_pub.publish(kidnap_msg);

                }




        this->world->SetPaused(false);

        //}//End of time constraints for what happens


}//end of timerCallbackMajorWithGlobalLocalization

void timerCallbackMajor(const ros::TimerEvent& event){
        ROS_INFO("INSIDE MAJOR EVENT TIMER CALLBACK!");

        callback_time = ros::Time::now();

        this->world->SetPaused(true);

        this->timer.stop();//so that it only fires once
```

```
        //Gazebo model pose:
        gazebo_msgs::GetModelState getmodelstate;
        getmodelstate.request.model_name = "mobile_base";
        this->Getclient.call(getmodelstate);

        current_x = getmodelstate.response.pose.position.x;
        current_y = getmodelstate.response.pose.position.y;
        current_z = getmodelstate.response.pose.position.z;

        tf::Quaternion q(getmodelstate.response.pose.orientation.x,
getmodelstate.response.pose.orientation.y, getmodelstate.response.pose.orientation.z,
getmodelstate.response.pose.orientation.w);
        tf::Matrix3x3 m(q);
        m.getRPY(roll, pitch, yaw);
        current_theta = yaw;




        x_prime = -1 ;//current_x +1000;//+ 0.2;
        y_prime =-2;//current_y+1000;//+0.2;
        theta_prime = current_theta;// + 0.5;//10deg = 0.17 rad

        //x_prime = 0;
        //y_prime = 0;
        //theta_prime = 3.14;



        newPose.position.x =x_prime;//prev_predicted_x; //_flat_tire;
        newPose.position.y =y_prime;//prev_predicted_y; //_flat_tire;



        //Unroll theta_prime into a quaternion
        tf::Quaternion quat;
        quat.setRPY(0, 0, theta_prime);//theta_prime);//prev_predicted_theta);//ROLL, PITCH
*YAW*

        newPose.orientation.x = quat.x();
        newPose.orientation.y = quat.y();
        newPose.orientation.z = quat.z();
        newPose.orientation.w = quat.w();
```

```
        //call gazebo set model state:
        gazebo_msgs::ModelState modelstate;
        modelstate.model_name = (std::string) "mobile_base";
        modelstate.reference_frame = (std::string) "world";
        modelstate.pose = newPose;
        gazebo_msgs::SetModelState setmodelstate;
        setmodelstate.request.model_state = modelstate;


        if (std::isnan(modelstate.pose.position.x) || std::isnan(modelstate.pose.position.y) ||
std::isnan(modelstate.pose.position.z) || std::isnan(modelstate.pose.orientation.x)
||std::isnan(modelstate.pose.orientation.y) || std::isnan(modelstate.pose.orientation.z) ||
std::isnan(modelstate.pose.orientation.w) ||std::isnan(modelstate.twist.linear.x) ||
std::isnan(modelstate.twist.linear.y) || std::isnan(modelstate.twist.linear.z)
||std::isnan(modelstate.twist.angular.x) || std::isnan(modelstate.twist.angular.y)
||std::isnan(modelstate.twist.angular.z)){
                ROS_INFO("There's a nan here...1");
                this->world->SetPaused(false);

                prev_x_2 = prev_x;
                prev_y_2 = prev_y;
                prev_theta_2 = prev_theta;
                prev_x = current_x;
                prev_y = current_y;
                prev_theta = current_theta;
                }

        else{

                //ROS_INFO("%f", ros::Time::now().toSec());
                this->client.call(setmodelstate);//coment out if you DO NOT want to PUBLISH



                std_srvs::Empty emptyArg;
//              ROS_INFO("Calling global localizaiton");
//              this->GlobalLocalizationClient.call(emptyArg);


                //ROS_INFO("%f", ros::Time::now().toSec());
                ROS_INFO("Changed X and y from %f , %f to %f , %f",current_x, current_y,
x_prime, y_prime);
                ROS_INFO("CHANGED THETA FROM %f to %f", current_theta, theta_prime);
```

```
                this->myGazeboFile<<callback_time<<std::endl;
                prev_x_2 = prev_x;
                prev_y_2 = prev_y;
                prev_theta_2 = prev_theta;
                prev_x = x_prime;//current_x;
                prev_y = y_prime;//current_y;
                prev_theta = theta_prime;//current_theta;

//Send a message that the event occurred.
                kidnap_msg.data = "KIDNAP OCCURRED.";
                ROS_INFO("SENDING MESSAGE");
                kidnap_pub.publish(kidnap_msg);

                }




        this->world->SetPaused(false);

        //}//End of time constraints for what happens


}//end of timerCallbackMajor



//This method collects data for evaluating the performance of the kidnapping mechanism.

void timerCallbackRelocationUnitTest(const ros::TimerEvent& event) {


        callback_time = ros::Time::now();

        this->world->SetPaused(true);

        //Gazebo model pose:
        gazebo_msgs::GetModelState getmodelstate;
        getmodelstate.request.model_name = "mobile_base";
        this->Getclient.call(getmodelstate);

        current_x = getmodelstate.response.pose.position.x;
```

```
        current_y = getmodelstate.response.pose.position.y;
        current_z = getmodelstate.response.pose.position.z;

        tf::Quaternion q(getmodelstate.response.pose.orientation.x,
getmodelstate.response.pose.orientation.y, getmodelstate.response.pose.orientation.z,
getmodelstate.response.pose.orientation.w);
        tf::Matrix3x3 m(q);
        m.getRPY(roll, pitch, yaw);
        current_theta = yaw;

        x_prime = 5.5;//+ 0.2;
        y_prime = 5.5;//+0.2;
        theta_prime = 0.5;//10deg = 0.17 rad


        newPose.position.x =x_prime;//prev_predicted_x; //_flat_tire;
        newPose.position.y =y_prime;//prev_predicted_y; //_flat_tire;



        //Unroll theta_prime into a quaternion
        tf::Quaternion quat;
        quat.setRPY(0, 0, theta_prime);//theta_prime);//prev_predicted_theta);//ROLL, PITCH
*YAW*

        newPose.orientation.x = quat.x();
        newPose.orientation.y = quat.y();
        newPose.orientation.z = quat.z();
        newPose.orientation.w = quat.w();
        float beforeMoveTime;


        //call gazebo set model state:
        gazebo_msgs::ModelState modelstate;
        modelstate.model_name = (std::string) "mobile_base";
        modelstate.reference_frame = (std::string) "world";
        modelstate.pose = newPose;
        gazebo_msgs::SetModelState setmodelstate;
        setmodelstate.request.model_state = modelstate;


        if (std::isnan(modelstate.pose.position.x) || std::isnan(modelstate.pose.position.y) ||
std::isnan(modelstate.pose.position.z) || std::isnan(modelstate.pose.orientation.x)
||std::isnan(modelstate.pose.orientation.y) || std::isnan(modelstate.pose.orientation.z) ||
std::isnan(modelstate.pose.orientation.w) ||std::isnan(modelstate.twist.linear.x) ||
std::isnan(modelstate.twist.linear.y) || std::isnan(modelstate.twist.linear.z)
```

```
||std::isnan(modelstate.twist.angular.x) || std::isnan(modelstate.twist.angular.y)
||std::isnan(modelstate.twist.angular.z)){
                ROS_INFO("There's a nan here...1");
                this->world->SetPaused(false);

                prev_x_2 = prev_x;
                prev_y_2 = prev_y;
                prev_theta_2 = prev_theta;
                prev_x = current_x;
                prev_y = current_y;
                prev_theta = current_theta;
                }

        else{

                //ROS_INFO("%f", ros::Time::now().toSec());
                beforeMoveTime = ros::Time::now().toSec();
                this->client.call(setmodelstate);//coment out if you DO NOT want to PUBLISH

        ROS_INFO("%f,%f,%f,%f,%f,%f,%f,%f",ros::Time::now().toSec(),ros::Time::now().toS
ec() - beforeMoveTime,current_x, current_y, current_z,x_prime,y_prime,theta_prime);

                //ROS_INFO("%f", ros::Time::now().toSec());

                this->myGazeboFile<<callback_time<<std::endl;
                prev_x_2 = prev_x;
                prev_y_2 = prev_y;
                prev_theta_2 = prev_theta;
                prev_x = x_prime;//current_x;
                prev_y = y_prime;//current_y;
                prev_theta = theta_prime;//current_theta;

//Send a message that the event occurred.
                kidnap_msg.data = "KIDNAP OCCURRED.";

                kidnap_pub.publish(kidnap_msg);

                }

        this->world->SetPaused(false);

        //}//End of time constraints for what happens


}//end of relocationUnitTest
```

```
std::vector<double> calculate_new_pose(double theta0, double x0, double y0, double theta1,
double x1, double y1, double theta2, double x2, double y2){


		double tx = x2, ty = y2, tth = theta2;


		//ROS_INFO("%f %f %f", theta0, theta1, theta2);

		std::vector<double> vintx_inty(2);

		std::vector<double> vxc_yc_mainradius(3);
		std::vector<double> vxc_yc_mainradiusTest;
		std::vector<double> vnewX_newY_newTheta(3);


		if((std::abs(theta0 - theta1) < .00174) || (std::abs(theta2 - theta1) < .00174)) {//
angles are close enough to assume straight line - 1.74e-6 rad <-> .1 deg
			ROS_INFO("**********Straight line!**********");

			vnewX_newY_newTheta[0] = (x2 -x1) + x2;
			vnewX_newY_newTheta[1] = (y2 - y1) + y2;
			vnewX_newY_newTheta[2] = theta2 + (theta2 -
theta1);//*(3.14/180);//keep it the same


			vnewX_newY_newTheta[0] = x1 + (x1-x0);
			vnewX_newY_newTheta[1] = y1 + (y1-y0);
			vnewX_newY_newTheta[2] = theta1 + (theta1-theta0);

			vnewX_newY_newTheta[0] = x2;
			vnewX_newY_newTheta[1] = y2;
			vnewX_newY_newTheta[2] = theta2;

			/*double radius = sqrt(pow(x2 - x1, 2) + pow(y2 - y1,2));
			vnewX_newY_newTheta[0] = x2 + radius *
cos(theta1+(180*(180/3.14)));
			vnewX_newY_newTheta[1] = y2 + radius *
sin(theta1+(180*(180/3.14)));//Tryin to do angle of prev+180 to get across the circle
			vnewX_newY_newTheta[2] = theta2 + (theta2-theta1);*/
```

```
		}
		else{


			//Logic expects degrees. Convert rads to degrees
			theta0=theta0*(180/3.14);
			theta1 = theta1 * (180/3.14);
			theta2 = theta2*(180/3.14);
			vintx_inty = find_intersection(theta1, x1, y1, theta2,  x2, y2);

			double intx = vintx_inty[0];
			double inty=vintx_inty[1];









			vxc_yc_mainradius=draw_circles(theta1,x1, y1, theta2, x2, y2, intx, inty);
//Orig method


			//Draw Arcs
			vnewX_newY_newTheta=draw_arcs(theta1, x1, y1, theta2, x2, y2,
vxc_yc_mainradius[0], vxc_yc_mainradius[1],vxc_yc_mainradius[2]);


			}



		return vnewX_newY_newTheta;
}



std::vector<double> draw_circles(double angleA, double posxA, double posyA, double
angleB,double posxB, double posyB, double angleC, double posxC, double posyC){
	//This is EB's version:
	double xc = 0.0;
	double yc = 0.0;
	double yDelta_a = posyB-posyA;
	double xDelta_a = posxB-posxA;
	double yDelta_b = posyC-posyB;
```

```
	double xDelta_b = posxC-posxB;


	double aSlope = yDelta_b/xDelta_a;
	double bSlope = yDelta_b/xDelta_b;

	double AB_Mid_x = (posxA+posxB)/2;
	double AB_Mid_y = (posyA+posyB)/2;

	double BC_Mid_x = (posxB+posxC)/2;
	double BC_Mid_y = (posyB+posyC)/2;

	if (yDelta_a ==0){	//aSlope == 0
		xc = AB_Mid_x;
		if (xDelta_b == 0){	//bSlope == Infinity
			yc = BC_Mid_y;
		}
		else{
			yc = BC_Mid_y + (BC_Mid_x - xc)/bSlope;
		}
	}
	else if (yDelta_b == 0){	//bSlope == 0
		xc = BC_Mid_x;
		if (xDelta_a ==0){	//aSlope == infinity
			yc = AB_Mid_y;
		}
		else {
			yc = AB_Mid_y + (AB_Mid_x - xc)/aSlope;
		}
	}

	else if (xDelta_a == 0){	//aSlope == infinity
		yc = AB_Mid_y;
		xc = bSlope*(BC_Mid_y -yc) + BC_Mid_x;
		}
	else if (xDelta_b == 0)		{	//bslope == infiinty
		yc = BC_Mid_y;
		xc = aSlope * (AB_Mid_y - yc) + AB_Mid_x;
		}
	else {
		xc = (aSlope * bSlope * (AB_Mid_y - BC_Mid_y) - aSlope * BC_Mid_x +
bSlope*AB_Mid_x)/(bSlope -aSlope);
		yc = AB_Mid_y - (xc - AB_Mid_x)/aSlope;
	}
```

```
        double main_radius = sqrt(pow(posxA - xc, 2) + pow(posyA - yc,2));

/*
        xc=intx;
        yc=inty;
        main_radius = sqrt(pow(posxB - xc,2) + pow(posyB-yc,2));
*/


/*
        std::vector<double> v(3);//= {xc, yc, main_radius};


        v[0] = xc;
        v[1] = yc;
        v[2] = main_radius;

*/
        double ret[] = {xc,yc, main_radius};
        std::vector<double> v(ret, ret+sizeof(ret)/sizeof(double));

        return v;

}




std::vector<double> find_intersection(double angle1, double posx1, double posy1, double
angle2,  double posx2, double posy2){

double ang1, ang2, px12, py12, px22, py22, x1, y1, x2, y2, x3, x4, y3, y4, den, intx, inty;
double pi=3.14;
//Calculate the angle at a 90degree angle to each point's angle
ang1 = angle1 + 90;
ang2 = angle2 + 90;


px12 = posx1 + 10 * cos(ang1*2*pi/360);


py12 = posy1 + 10 * sin(ang1*2*pi/360);

px22 = posx2 + 10 * cos(ang2*2*pi/360);
```

```
py22 = posy2 + 10 * sin(ang2*2*pi/360);



x1 = posx1;
y1 = posy1;
x2 = px12;
y2 = py12;

x3 = posx2;
y3 = posy2;
x4 = px22;
y4 = py22;

//Calculating the _____. No idea.
den = ((x1-x2)*(y3-y4) - (y1-y2)*(x3-x4));



//if den is not 0
if (std::abs(den) > .0001){
   intx = ((x1*y2 - y1*x2)*(x3-x4) - (x1-x2)*(x3*y4 - y3*x4)) / den;
   inty = ((x1*y2 - y1*x2)*(y3-y4) - (y1-y2)*(x3*y4 - y3*x4)) / den;

}
else{
        intx = std::numeric_limits<double>::quiet_NaN();
        inty = std::numeric_limits<double>::quiet_NaN();
}



std::vector<double> v={intx,inty};
//v[0] = intx;
//v[1] = inty;

return v;

}//end of find_intersection

std::vector<double> draw_circles(double angle1,double posx1, double posy1, double angle2,
double posx2, double posy2, double intx, double inty){
```

```
//double intx=5;
//double inty=5;

//double* pointer;
double radius, d, a, h, x0, y0, x1, y1, x2, y2, x3_1, y3_1, x3_2, y3_2, xc_a, yc_a, xc_b, yc_b,
dist_a, dist_b, xc, yc,main_radius;

//Calculate teh radius based on the distance formula between teh 2 points
radius = sqrt(pow(posx1 - posx2, 2) + pow(posy1 - posy2,2));
//find crossing points
d = sqrt(pow(posx1 - posx2, 2) + pow(posy1 - posy2, 2));
x0 = posx1;
y0 = posy1;
x1 = posx2;
y1 = posy2;


//middle point between the x's
x2 = (x0 + x1)/2;                   //Interested in the +'s

//middle point between the y's
y2 = (y0 + y1)/2;

//half of the radius
a = d/2;

//distance? btw radius and half the radius
h = sqrt(pow(radius, 2) - pow(a,2));

//midpoint of x's + dist btw radius and midpoint * midpoint of y's, divided by radius?

//Is this chord-y stuff?

x3_1 = x2 + h * ( y1 - y0 ) / d;
y3_1 = y2 - h * ( x1 - x0 ) / d;

x3_2 = x2 - h * ( y1 - y0 ) / d; //minusing the h*... this time. Was prev. +
y3_2 = y2 + h * ( x1 - x0 ) / d;


xc_a = (x3_1 + intx)/2;
yc_a = (y3_1 + inty)/2;

dist_a = sqrt(pow(x3_1 - xc_a, 2) + pow(y3_1 - yc_a, 2));

xc_b = (x3_2 + intx)/2;
```

```
yc_b = (y3_2 + inty)/2;

dist_b = sqrt(pow(x3_2 - xc_b,2) + pow(y3_2 - yc_b,2));

if (dist_a < dist_b){
   xc = xc_a;
   yc = yc_a;
}
else{
   xc = xc_b;
   yc = yc_b;
}

main_radius = sqrt(pow(posx1 - xc,2) + pow(posy1 - yc,2));


xc=intx;
yc=inty;
main_radius = sqrt(pow(posx1 - xc,2) + pow(posy1-yc,2));

//std::vector<double> v = {xc, yc, main_radius};
/*v[0] = xc;
v[1] = yc;
v[2] = main_radius;
*/

double ret[] = {xc,yc, main_radius};
std::vector<double> v(ret, ret+sizeof(ret)/sizeof(double));

return v;
}



std::vector<double> draw_arcs(double angle1, double posx1, double posy1, double angle2,
double posx2, double posy2, double centX, double centY, double radius){

double* pointer;
double projX1, projY1, projX2, projY2, ang1, ang2, px, py, dir, arc_deg, arc_len, new_rad,
tmpang, new_arc_deg,new_centX, new_centY, newX, newY, newTheta, s1, s2;
double pi = 3.14;
projX1 = posx1 - centX;
projY1 = posy1 - centY;
projX2 = posx2 - centX;
projY2 = posy2 - centY;
```

```
ang1 = atan2(projY1,projX1);
ang2 = atan2(projY2,projX2);

//force angle to 0-2pi
if (ang1 < 0){
   ang1 = ang1 + 2 * pi;
}

if (ang2 < 0){
   ang2 = ang2 + 2 * pi;
}
/*
% use cross product to determine if the arc opens counter clockwise or

% clockwise
% a x b = ||a|| ||b|| sin(theta) n
% a x b = [(i j k); (u1 u2 u3); (v1 v2 v3)];
% a x b = [(i j k); (centX-posx1 centY-posy1 0); (1, tan(angle1) 0)]
% a x b = posx1*pyk + posy1*px*(-k) % with above zeroz factored in

*/

px = cos(angle1*2*pi/360);
py = sin(angle1*2*pi/360);
dir = (centX-posx1)*py - (centY-posy1)*px; //in the k direction

if  (dir < 0){
   arc_deg = ang2 - ang1;

}
else{
   arc_deg = ang1 - ang2;

}

if (arc_deg < 0){
   arc_deg = arc_deg + 2*pi;
}
if (arc_deg > 2*pi){
   arc_deg = arc_deg - 2*pi;
}

//% arc length
arc_len = arc_deg/(2*pi) * radius;

//% radius = (rad_right + rad_left) / 2;
```

```
//% radius ~ wheel_speed_right / wheel_speed_left;
//% 10% change in one wheel speed results in 10% change in radius                    //Are we sure it's a 10% change in raidus? or is it 5?

//%based on direction
//%negative dir means turning left
//%positive dir means turning right
new_rad = radius * 0.8;//Change radius here!  This determines the severity of the kidnapping or the wheel slip event and symbolizes the reduction of the motion achieved
new_arc_deg = arc_len * (2*pi) / (new_rad);

tmpang = atan2((centY-posy1),(centX-posx1));
new_centX = centX + (new_rad-radius)*cos(tmpang);
new_centY = centY + (new_rad-radius)*sin(tmpang);


/*
%there is lots of room to play with the above math
%depending on if the change is multiplicative or additive
*/

//%new robot pos
if  (dir < 0){
   newX = new_centX+new_rad*cos(ang1+new_arc_deg);
   newY = new_centY+new_rad*sin(ang1+new_arc_deg);
   newTheta = angle1*(2*pi/360) + new_arc_deg;
}
else{
   newX = new_centX+new_rad*cos(ang1-new_arc_deg);
   newY = new_centY+new_rad*sin(ang1-new_arc_deg);
   newTheta = angle1*(2*pi/360) - new_arc_deg;
}
// the original angle change + the change in angle change

s1 = tan(angle1*2*pi/360); //What am I missing with this 2*pi/360?

s2 =  tan(angle2*2*pi/360);

//find points to plot

//Question: why are plots to point different than newX, newY?
px = newX + 2 * cos(newTheta);
py = newY + 2 * sin(newTheta);


//std::vector<double> v = {newX, newY, newTheta};
```

```
/*
v[0] = newX;
v[1] = newY;
v[2] = newTheta;
*/


double ret[] = {newX,newY, newTheta};
std::vector<double> v(ret, ret+sizeof(ret)/sizeof(double));

return v;

}

private: transport::NodePtr node;
private: transport::SubscriberPtr sub;



};//end class



//register plugin
GZ_REGISTER_WORLD_PLUGIN(CallSpawnDeleteWorldPlugin)
}//end namespace
//END OF PLUGIN
```

Python Service for Listening to ROS Messages and Classifying for Kidnap Detection

```
#!/usr/bin/env python
# Software License Agreement (BSD License)
#
# Copyright (c) 2008, Willow Garage, Inc.
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
#  * Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#  * Redistributions in binary form must reproduce the above
#    copyright notice, this list of conditions and the following
#    disclaimer in the documentation and/or other materials provided
#    with the distribution.
```

```
#  * Neither the name of Willow Garage, Inc. nor the names of its
#    contributors may be used to endorse or promote products derived
#    from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.
#:
# Revision $Id$


import rospy
from amcl.msg import *
from std_srvs.srv import Empty
import matplotlib.pyplot as pl
import numpy as np
import csv
import array
import os
import sys



#THIS WILL ONLY RUN IF YOU USE THE 'SOURCE' COMMAND IN THE CONSOLE
WINDOW BEFORE RUNNING THIS CODE because of the dependency on the
filter_covariance_msg message type
#source /home/ebrenna8/amcl_overlay/devel/setup.bash

#The error you'll receive is "ImportError: No module named msg"

#define the threshold for considering the NN output a kidnapping indicator
kidnapping_threshold_value = 1.2
input_ps_ystd=1 #always 1
input_ps_ymean = 0 #always 0
```

```
global global_localization_counter
global_localization_counter = 0
global do_not_call
do_not_call = False
global prev1
global prev2
global prev3
global prev4
prev1=False
prev2=False
prev3=False
prev4=False


# Constants used in the neural net (Major)
#Input 1
x1_step1_xoffset =[[-0.417395479043843],[-0.665118718889251],[-0.779427359138429],[-
0.861850936207229],[-0.987811093989633],[-0.631616608173962],[-1.28952929514543],[-
1.40300213812519],[-1.32775102171854],[-0.924209297577193]]

x1_step1_gain =
[[0.300138588647975],[0.238547613084996],[0.259733040892764],[0.206763366149844],[0.2
43186543880447],[0.515910780298467],[0.248302846118538],[0.321464093671442],[0.30548
8144998311],[0.19468106000605]]
x1_step1_ymin = -1

# Layer 1
b1 = [[-1.9892537431924884],[1.9336731093776252],[1.2983535583428367],[-
0.092002546611450464],[0.68125742685358648],[-
0.10620095722291445],[0.81764632828410111],[-0.30803683621605638],[-
0.64951802159300398],[-
0.30890651057687374],[0.75354235619256038],[1.5893161112280159],[1.8977125968228377
],[-1.7236819257891789],[2.112376881750536]]

IW1_1=[[0.21797726281083721,0.39016342840527446,-0.11010938719989172,-
0.89730071914422671,0.25927043982094788,-0.27175491510430561,-1.0302432663473633,-
0.55097148721644296,-0.84956377314308573,-0.082102749362913841],[-
2.3955738185889222,0.40401160712035455,0.61493581954492438,0.83073973236364218,0.9
2552710680441408,-0.2379324368151067,0.28530035457980712,-
0.94034773140547012,0.51893353296928313,-0.52377657249119769],[-
0.79156628997615108,-0.76490370936924723,0.07029131479765241,-0.54826658807540951,-
0.64965947370626609,-0.16005427938747269,-
0.058953701383777958,0.90141252300968533,-0.22303209710949051,-
0.73497837592504722],[0.14394062454785236,-0.24115325216875466,-
0.37369224873429746,-0.16269801091829564,0.076716579020957493,-
0.86529169263327566,-1.3956941523302775,-0.78936486018347773,-
```

0.79967424483310989,0.83330643447562369],[0.0066931521936632599,-0.037189679148745658,0.77552543416455555,-0.22958521360446868,-0.43192817363337344,-0.70537539811246508,0.08477270907136028,-0.58050299030582797,-0.19536910396386442,-1.409969119267088],[0.22827688758657985,0.20574070812093839,1.1811234402698316,1.074641625760582,-1.0861604093674213,0.57761802002475737,-1.1074126626386533,1.267694617342132,-0.2900909114715845,1.0470807339447634],[-0.34412587383163956,0.47556086395799935,2.376173056976961,-2.1863163822681857,-1.2513839633014245,0.2171321594728686,-0.38536413901716177,-0.11162222413477083,0.50448072760838847,-0.41188064054145845],[0.32709476186437231,0.11119195232747774,0.023704693334010311,-0.72344160269584279,-0.77072361151944224,-1.2906350454533584,0.83208246677163267,0.035173216061224664,-0.2007108747332266,-0.26161671334792158],[0.13006592009812221,-0.21873744039390386,1.0350164971037334,0.78731500888836026,-0.44668255594546663,0.48245250004456486,0.077675698453573055,1.0202241350408741,0.26737738596156357,-0.85005461372453273],[-1.1048374629789826,-0.96489549109395911,0.69724449282241319,-0.58434428466902932,0.14501691913614978,0.43579155970250821,-1.0193245574096175,-0.47192366629108623,0.13151376734052664,0.18570711100210455],[0.75418439068702559,-0.68919320931079919,0.07837664176702594,-0.22443119848322315,-0.49583582708866647,-0.95158994331432789,-0.493526056648671,-0.097673841360452804,0.70055408670631236,-0.74572523206062236],[-1.3508454687583022,0.35908685206256535,-0.34955520873958956,0.49709815469609087,0.80356130553608662,-1.5471875243375426,-0.45728698326820055,0.15470675045755017,0.054613462584038472,2.6552137029292244],[-1.5442735679655355,0.46450772122772405,-0.7476140739191689,-0.69604588006676271,-0.14700420665740957,2.2159156979568295,1.0348397995264047,-1.2197051619033736,-1.7345078470058364,2.1899798294567518],[-0.15001493092222692,-0.27674708708497286,-0.82320409117982574,-0.26794651023149507,0.82358156828262497,1.4791498225576483,0.13531053516695901,0.6167976255702583,0.18299875416843864,-0.14847258684157311],[0.051109755367930193,0.17350166821241977,0.018493167283935182,0.22870917505639302,-0.23385677828470403,-0.19413742397423389,0.76979600924495273,-0.10435717183489743,-1.1622434019750933,0.59275921170739831]]

# Layer 2
b2 = 0.25683677953871653
LW2_1=[0.29417987628437353,1.5152829890431485,0.14185972965534677,0.001768380387405951,-0.21345443607522935,1.0186135053712448,-1.2486752197615534,1.3808966956967981,-0.070525731321636337,1.7388789545466712,-0.14413591149003968,-1.6516450554508446,-1.5342058174921351,0.34653078366841106,0.048280636167513172]

Output 1

```
y1_step1_ymin = -1
y1_step1_gain = 0.845883445765254
y1_step1_xoffset = -0.551215893485094

#input_ps_xstd
input_ps_xstd=[[134.6122],[25.1668],[26.6226],[31.1179],[32.5870],[146.2607],[35.5073],[38.5
757],[53.1549],[112.7203]]
input_ps_xmean=[[56.1865],[   16.7389],[   20.7504],[   26.8190],[   32.1898],[   92.3807],[
45.7877],[   54.1218],[   70.5765],[  104.1772]]


#define values for neural network typpes
def loadMinorXY():
        input_ps_xmean = [[17.6965],[   14.6560],[   19.6899],[   26.4802],[   33.8487],[
72.7658],[   51.5014],[   62.9844],[   86.2323],[  133.0910]]
        input_ps_xstd = [[61.8656],[   22.7066],[   23.3188],[   27.0519],[   30.1989],[
113.9274],[   29.5977],[   31.3939],[   45.1031],[  103.9875]]
        #% Output 1
        y1_step1_ymin = -1
        y1_step1_gain = 0.93424418852899
        y1_step1_xoffset = -0.688031918010382
        #% Layer 2
        b2 = 0.34186571475312466
        LW2_1=[-0.66231906417810504,-0.40173891618368501,-
0.18528733383650284,0.054618744783013561,0.37540953615793976,0.47044867107686827,-
0.65228648698760971,0.11446583571033581,0.69935727383272284,-
1.0299459917846046,0.94293086169263796,1.2481704193669576,-0.39420797786431488,-
0.41002535348284741,-1.0521169937740877];
        #Layer 1
        IW1_1=[[-0.9467535311497014,-0.89176278743242066,0.48393639866447813,-
0.078993775901212607,0.16512091134320731,0.33267438792136123,-
0.64830467097142674,-0.75887478595245705,0.36292088580932486,-
0.67118342473406056],[-0.52961447648736992,-
0.8659936077951812,0.339898719007386,0.01068741055009343,0.27635817125409701,-
0.079843755488573115,-0.73168921445520685,0.57789000661078771,-1.0548445065077796,-
0.59513791621757095],[-0.20112268916494247,0.75908909013271786,-
0.50024722549437395,-0.43521145956564727,-0.28549189485880827,-
0.91883683615982303,-0.4088826211419872,-0.86179502943005881,-
0.66093125835986599,0.063350386436714312],[-0.98878749171925162,-
1.068060551218053,0.21753830673707378,-
0.71885216662434936,0.14835870919962088,0.060274316631352534,0.80021003189551376,0
.044475569744135253,0.1329531267452646,-
0.042238148779471678],[0.88219640067266658,-0.46934766948019041,-
0.49361662537647066,-0.92242211012762887,0.24905266569169673,-1.3991052312229955,-
1.3377266211349568,0.28202837565335676,0.82648299879583897,-0.37896892538883059],[-
1.3456113365456399,-0.092512965552705981,-
```

0.098058288413125472,0.03696578251291617,0.1925062908515191,0.069277151718791188,1
.4309192683515488,0.53840431929598997,-0.21054837828117856,-0.30602198926404245],[-
0.73598769278714138,-0.5480554627507721,0.21394214895383876,-0.8336960119976724,-
0.006868710471661713,-0.78789839632026737,0.71495940741728747,-
0.59411385614314116,0.64896160858486607,-
0.52704210693135678],[0.26141316712260537,-
0.46747135875882129,1.0407324211352513,0.59300122141887979,-0.82025985286492054,-
0.030057705269708027,0.66214278863365794,0.80402077699681851,0.12212170633303052,-
0.19741059980427092],[-
0.020359094928403688,0.75722686780569293,0.41674162606353815,0.12471845122408351,-
0.18306235596593309,0.82438470783707762,-
1.2744803025510012,0.99506751905562518,0.17935253962886982,-0.08202864430374214],[-
1.1966178987352543,0.81269677037531951,1.1000606772002162,-0.12699857109527998,-
0.39117877737991591,-1.4257808776925627,0.37140866836940162,1.4490656672973445,-
0.40840251794099125,0.33767023775813981],[-
0.0093093772980819417,1.3643530310405332,-0.44150884955310477,-
0.78900221523712888,-0.93835088482694262,-
1.5748950718870995,1.0781009191708877,1.089978222605631 8,1.2543357890365632,-
1.738329916147731],[-0.37409554658732724,-
1.5178247321635936,0.0053711313562620315,0.28631852668909114,-
0.076966895345868314,0.25411902695247457,1.5035192543006068,0.37730979175794388,-
1.4033210434445715,1.2803776672195695],[0.85099029453121156,-0.47947115612507152,-
0.11911958148017709,-0.78380620083611241,-
0.058857540397335206,0.15680368907887066,0.96000469762592189,-
0.36895862608578622,-0.8366141137787374,0.7246757169697543],[-
0.074808279358415761,-0.50339302142123521,0.64625737152487495,-
0.16072830627975898,0.21364526053551036,0.43426795073727359,0.22527269704702535,0.
34742620723649031,-0.70035971035715983,0.88541922094029502],[0.18453833767646508,-
0.74707230467733643,-0.92015286681119135,0.20733400081769482,0.62218249467549391,-
0.024623277767324359,0.74382174841574122,0.71851589192293375,0.85266107003634295,0
.1568426577535568]]
b1 =
[[1.9407318311286064],[1.7849376949311386],[1.2789616446562662],[0.99611344209948161
],[-0.43199383322883345],[0.50137282211324941],[0.16893212751822545],[-
0.15603194422694835],[0.33376756254733114],[-0.58035114895789852],[-
0.47076044630280361],[0.96507988205508211],[1.2875467256861699],[-
1.6751523247913855],[1.9251669634866886]]
#Input 1
x1_step1_xoffset = [[-0.286047913962952],[-0.645450688992174],[-
0.844381134185687],[-0.97886724941025],[-1.12085947107572],[-0.638703108074877],[-
1.74005068046043],[-2.00626210761253],[-1.91189355974711],[-1.27987402844859]]
x1_step1_gain =
[[0.13793880534161],[0.215228352389046],[0.24546055874778],[0.193920345211168],[0.225
364955563111],[0.454800015741184],[0.230331918860386],[0.261616068651488],[0.2592132
23133741],[0.179598514955609]]
x1_step1_ymin = -1

```
	kidnapping_threshold_value =0.4
	return
kidnapping_threshold_value,x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2
_1,y1_step1_ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean



def loadMajor():
	#define the threshold for considering the NN output a kidnapping indicator

	kidnapping_threshold_value = 1.2

	#Input 1
	x1_step1_xoffset =[[-0.417395479043843],[-0.665118718889251],[-
0.779427359138429],[-0.861850936207229],[-0.987811093989633],[-0.631616608173962],[-
1.28952929514543],[-1.40300213812519],[-1.32775102171854],[-0.924209297577193]]

	x1_step1_gain =
[[0.300138588647975],[0.238547613084996],[0.259733040892764],[0.206763366149844],[0.2
43186543880447],[0.515910780298467],[0.248302846118538],[0.321464093671442],[0.30548
8144998311],[0.19468106000605]]
	x1_step1_ymin = -1

	# Layer 1
	b1 = [[-1.9892537431924884],[1.9336731093776252],[1.2983535583428367],[-
0.092002546611450464],[0.68125742685358648],[-
0.10620095722291445],[0.81764632828410111],[-0.30803683621605638],[-
0.64951802159300398],[-
0.30890651057687374],[0.75354235619256038],[1.5893161112280159],[1.8977125968228377
],[-1.7236819257891789],[2.112376881750536]]

	IW1_1=[[0.21797726281083721,0.39016342840527446,-0.11010938719989172,-
0.89730071914422671,0.25927043982094788,-0.27175491510430561,-1.0302432663473633,-
0.55097148721644296,-0.84956377314308573,-0.08210274936291384 1],[-
2.3955738185889222,0.40401160712035455,0.61493581954492438,0.83073973236364218,0.9
2552710680441408,-0.2379324368151067,0.28530035457980712,-
0.94034773140547012,0.51893353296928313,-0.52377657249119769],[-
0.79156628997615108,-0.76490370936924723,0.07029131479765241,-0.54826658807540951,-
0.64965947370626609,-0.16005427938747269,-
0.058953701383777958,0.90141252300968533,-0.22303209710949051,-
0.73497837592504722],[0.14394062454785236,-0.24115325216875466,-
0.37369224873429746,-0.16269801091829564,0.076716579020957493,-
0.86529169263327566,-1.3956941523302775,-0.78936486018347773,-
0.79967424483310989,0.83330643447562369],[0.0066931521936632599,-
0.037189679148745658,0.77552543416455555,-0.22958521360446868,-
0.43192817363337344,-0.70537539811246508,0.08477270907136028,-0.58050299030582797,-
```

0.19536910396386442,-
1.409969119267088],[0.22827688758657985,0.20574070812093839,1.1811234402698316,1.07
4641625760582,-1.0861604093674213,0.57761802002475737,-
1.1074126626386533,1.267694617342132,-0.2900909114715845,1.0470807339447634],[-
0.34412587383163956,0.47556086395799935,2.376173056976961,-2.1863163822681857,-
1.2513839633014245,0.2171321594728686,-0.38536413901716177,-
0.11162222413477083,0.50448072760838847,-
0.41188064054145845],[0.32709476186437231,0.11119195232747774,0.023704693334010311
,-0.72344160269584279,-0.77072361151944224,-
1.2906350454533584,0.83208246677163267,0.035173216061224664,-0.2007108747332266,-
0.26161671334792158],[0.13006592009812221,-
0.21873744039390386,1.0350164971037334,0.78731500888836026,-
0.44668255594546663,0.48245250004456486,0.077675698453573055,1.0202241350408741,0.
26737738596156357,-0.85005461372453273],[-1.1048374629789826,-
0.96489549109395911,0.69724449282241319,-
0.58434428466902932,0.14501691913614978,0.43579155970250821,-1.0193245574096175,-
0.47192366629108623,0.13151376734052664,0.18570711100210455],[0.75418439068702559,-
0.68919320931079919,0.07837664176702594,-0.22443119848322315,-0.49583582708866647,-
0.95158994331432789,-0.493526056648671,-0.097673841360452804,0.70055408670631236,-
0.74572523206062236],[-1.3508454687583022,0.35908685206256535,-
0.34955520873958956,0.49709815469609087,0.80356130553608662,-1.5471875243375426,-
0.45728698326820055,0.15470675045755017,0.054613462584038472,2.6552137029292244],[-
1.5442735679655355,0.46450772122772405,-0.7476140739191689,-0.69604588006676271,-
0.14700420665740957,2.2159156979568295,1.0348397995264047,-1.2197051619033736,-
1.7345078470058364,2.1899798294567518],[-0.15001493092222692,-0.27674708708497286,-
0.82320409117982574,-
0.26794651023149507,0.82358156828262497,1.4791498225576483,0.13531053516695901,0.6
167976255702583,0.18299875416843864,-
0.14847258684157311],[0.051109755367930193,0.17350166821241977,0.018493167283935182
,0.22870917505639302,-0.23385677828470403,-
0.19413742397423389,0.76979600924495273,-0.10435717183489743,-
1.1622434019750933,0.59275921170739831]]

# Layer 2
b2 = 0.25683677953871653
LW2_1=[0.29417987628437353,1.5152829890431485,0.14185972965534677,0.001768
380387405951,-0.21345443607522935,1.0186135053712448,-
1.2486752197615534,1.3808966956967981,-0.070525731321636337,1.7388789545466712,-
0.14413591149003968,-1.6516450554508446,-
1.5342058174921351,0.34653078366841106,0.048280636167513172]

# Output 1
y1_step1_ymin = -1
y1_step1_gain = 0.845883445765254
y1_step1_xoffset = -0.551215893485094

```
        #input_ps_xstd
        input_ps_xstd=[[134.6122],[25.1668],[26.6226],[31.1179],[32.5870],[146.2607],[35.507
3],[38.5757],[53.1549],[112.7203]]
        input_ps_xmean=[[56.1865],[   16.7389],[   20.7504],[   26.8190],[   32.1898],[
92.3807],[   45.7877],[   54.1218],[   70.5765],[  104.1772]]

        return
kidnapping_threshold_value,x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2
_1,y1_step1_ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean



def loadMinorTheta():
        input_ps_xmean = [[59.7876],[   22.6389],[   26.3573],[   31.0412],[   36.4220],[
88.4648],[   46.5109],[   53.1709],[   66.0387],[   94.2124]]
        input_ps_xstd=[[133.1591],[   33.1023],[   33.1126],[   33.4851],[   35.9241],[
139.0877],[   36.0335],[   38.9130],[   52.5266],[  106.3781]]
        #Input 1
        x1_step1_xoffset = [[-0.448993367030028],[-0.68390783560027],[-
0.795992247606312],[-0.927015704899677],[-1.01385950074979],[-0.636036226758363],[-
1.29076983465399],[-1.36640366324522],[-1.25724276862478],[-0.885637202872422]]
        x1_step1_gain =
[[0.296898844405782],[0.192455434220275],[0.203769659102015],[0.238327773179083],[0.2
57520389557107],[0.404913174744542],[0.280416071120398],[0.324274917390756],[0.30187
706197263],[0.183727347262379]]
        x1_step1_ymin = -1
        #Layer 1
        b1 = [[1.8354563291838815],[-1.5847709141205677],[-1.2077592484302049],[-
1.0482337754812214],[-1.0146924643094961],[-0.99846411048835693],[-
0.45589929092846543],[-0.15505789147704543],[0.14649954845448246],[-
1.0466410642236015],[-1.1341967760840517],[0.88636870292432246],[-
1.3189740060633257],[1.559667768336783],[1.6128664459439983]]
        IW1_1=[[-0.53633250057303972,-0.76702794016547327,-0.96082470683688004,-
0.57725137479147426,-0.71946709112815799,-
0.3464004516457671,0.68995499093376778,0.066642516112380534,-0.27267191768993282,-
0.20922814938547088],[0.74255222980502167,0.50793442732820537,-
0.55397250079845795,0.8275026280317449,-
0.048488238439949705,0.040256922112986544,0.8531289927587683,-
0.83458500772732991,-
0.32058227379597504,0.23330483725738504],[0.44048389701754809,0.099323936720236733
,0.90017122804004313,0.98972314589584964,0.75588162990304397,-0.50173083275253472,-
0.67614779600763508,-1.1425904357231222,-0.010048214719364214,-
0.045521267558546338],[0.11528950458376157,-
0.0063040375702511295,0.77889542275266266,0.28498138255247174,0.060290623705143015
,-0.95316192425406054,0.15494889329508979,0.77615679868018272,-
```

0.76062320651277127,0.74437740936222896],[0.32788041993853401,0.69675038951627877,-1.008968373304056,0.41973114032063125,-0.71535117634924739,0.35454741148642832,-0.44493677789482577,-0.0056439053357651425,-0.70205961230378622,-0.51798594428487543],[-0.43240606034847762,-1.6212519580398654,-0.53398870355920092,0.20031949062481327,0.40918948251452464,-0.54779964325196973,0.030477856108901871,0.69555288635508883,0.20419506137401891,-0.22810422025152347],[0.64969722227106963,0.0013024859614275998,0.62218249612245802,-0.48395750635392171,0.97771647022654762,0.2063237616305662,0.29127373882902802,-0.044530137530710071,-0.23982616576189814,-2.4052432672666337],[-0.2653103266884036,0.24310452704189234,0.88949704224445281,-1.0293975783090583,-0.028431382632735154,-0.12213442847817274,0.52659119021408229,0.8527756764706893,-0.053766952249768268,-0.8526795284187203],[-0.043173455988279812,0.54832020770794798,1.5143422882553574,0.86066340926405571,-0.97482956132193221,0.2014199658282127,-0.10806963592134008,0.047175282338485384,-0.28082206575039903,0.44984127104786215],[0.11808445500396528,-0.074118517400896158,0.44378769524915868,-0.61266072629053403,0.8829754101384738,-0.5415485206268319,-0.64926570809875606,-0.76743349701697905,0.93272391026601753,-0.33984071751314798],[-0.25753914536278283,0.868165593218721,0.67842880870936972,0.27013681311115928,-0.18031484475790535,0.76797641575430486,-0.057792627846204619,-0.2881763128318014,0.62507946338970211,0.988761609115853 2],[0.71026938568315034,-1.0308863027794377,-0.096222928126841578,0.89186561505827033,1.3221433769172617,-0.41562451844471088,-0.17018131584835583,-0.5187297366876521,-0.05476023007573539,-0.33898094903656989],[-0.78565081060565845,-0.33229776236054726,-0.38442294944784544,1.1560887404166922,0.34715390382523148,0.64843282136684144,-0.034841008997605387,0.64868751012867143,-0.35745522376268857,0.27070754950099807],[0.27565161015584944,0.22221592638337065,-0.00052505221983465203,0.24355823884345237,-0.95758939425769296,-0.78000411905941514,-0.3269431578158305,-0.7318079380458713,-0.70855741719055065,0.51276987094006543],[0.87843839032958837,-0.57396339144800712,0.092357093642757238,-0.73069793313408316,-0.67306510586556023,-0.57448866150078859,0.63941827007611607,-0.45331697885815875,0.92360645052968771,0.83908304629550312]]

# Layer 2
b2 = -0.053515373141880561
LW2_1=[0.00014005447982335523,-0.12813282688592859,1.1092626453857839,0.045410277960576458,1.0465337702810356,0.85972270787859961,1.0961465852438197,-0.14128777831485459,0.39489713875798244,0.61605112436210807,0.38547046469386903,-0.9400933655932876,-0.7573763233568882,0.49421508198039588,0.90501592158479849];
# Output 1
y1_step1_ymin = -1
y1_step1_gain = 0.990300579282151
y1_step1_xoffset = -0.868032421118075
kidnapping_threshold_value =0.15

```
        return
kidnapping_threshold_value,x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2
_1,y1_step1_ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean



def tansig(x):
   return 2/(1+np.exp(-2*x)) - 1 #Per
https://www.mathworks.com/help/deeplearning/ref/tansig.html

def sigmoid(x):
   return 1/(1+np.exp(-x))

def minus(x, settings):
        return x - settings

def times(x, settings):
        return np.multiply(x, settings)

def plus(x, settings):
        return x + settings
def divide(x,settings):
        return x/settings
def divideInt(x,settings):
        return np.divide(x,settings)
def mapminmax_apply(x,settings_gain,settings_xoffset,settings_ymin):
        y = minus(x,settings_xoffset)
        y = times(y,settings_gain)
        y = plus(y,settings_ymin)
        return y
def mapminmax_reverse(x,settings_gain,settings_xoffset,settings_ymin):
        y = minus(x,settings_ymin)
        y =divide(y,settings_gain)
        y = plus(y,settings_xoffset)
        return y

def mapstd_apply(x,input_ps_xmean,input_ps_ystd,input_ps_xstd,input_ps_ymean):
        y = minus(x,input_ps_xmean)
        y2 = divideInt(input_ps_ystd,input_ps_xstd)
        y = times(y,y2)
        y = plus(y,input_ps_ymean)
        return y
```

```
def
applyNN(input,x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2_1,y1_step1_
ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean):

        return_array= []

        for X in input:
                #X = [2,         12,      15,      9,       18,      40,      93,      94,      92,      126]

                X = np.transpose(X)
                X = np.array(X)[np.newaxis].T #so that it's a 10x? array

                X = np.array(X).reshape((-1, 1)) # This line is for converting X into nx1 2D
matrix.

                TS = X.shape[1] #size(X,2)


                Q = X.shape[0] #size(X{1},2)
                Y = [] #Y should be 1xTS for the outputs


        X=mapstd_apply(X,input_ps_xmean,input_ps_ystd,input_ps_xstd,input_ps_ymean) #this
step is pre-processing that I did in MATLAB outside of the neural network.  Mapstd was applied
as pre-processing for all inputs in the training and testing sets.

                for ts in range(0,TS):
                        # Input 1
                        Xp1 =mapminmax_apply(X,x1_step1_gain,
x1_step1_xoffset,x1_step1_ymin)

                        # Layer 1
                        a1 = tansig(b1 + np.dot(IW1_1 , Xp1))

                        # Layer 2
                        a2 = b2 + np.dot(LW2_1,a1)

                        # Output 1
                        return
mapminmax_reverse(a2,y1_step1_gain,y1_step1_xoffset,y1_step1_ymin)


def update_sliding_window(tf):
   global prev1
   global prev2
   global prev3
```

```
    global prev4

    print("Sliding window currently has the following data for previous timesteps:")
    print("Current Timestep (t) has Kidnap data: " + str(tf))
    print("t-1 Timestep has Kidnap data: " + str(prev1))
    print("t-2 Timestep has Kidnap data: " + str(prev2))
    print("t-3 Timestep has Kidnap data: " + str(prev3))
    print("t-4 Timestep has Kidnap data: " + str(prev4))

    if (prev1 and prev2 and prev3 and prev4 and tf):
            print("5 consecutive kidnap events were detected. Prolonged disturbance kidnapping is
likely. Consider a new driving strategy.")
    prev1 = tf
    prev2 = prev1
    prev3 = prev2
    prev4 = prev3

def global_localization():
    global global_localization_counter #used to determine when the last call to global_localization
was
    global do_not_call #used to prevent calling global_localization too oftenn
    if global_localization_counter > 10:
        global_localization_counter = 0
        do_not_call = False

    if (do_not_call == False):
                print("Calling amcl global_localization service with rospy")
                rospy.wait_for_service('global_localization')
                global_localization = rospy.ServiceProxy('global_localization', Empty)
                resp = global_localization()
                print("Distributed particles for global localization. ")
                do_not_call = True #don't call global_localization consecutively, because it takes time
to take effect
    else:
                print("Refraining from calling global_localization service because a recent call to
global_localization is still completing")
                print("It has been " + str(global_localization_counter) + " timesteps since the most
recent call to the global_localization service")

    global_localization_counter = global_localization_counter + 1


def callback(data):

    #rospy.loginfo(rospy.get_caller_id()+"I heard %s",data.cloud_weights)
```

```
    kidnap_detected = False

    #Shape into a histogram
    #bin_heights, bin_boundaries, patches = pl.hist(np.asarray(data.cov), 10)

    hist_output = np.histogram(np.asarray(data.cov), bins=10)
    bin_heights = hist_output[0].ravel()
    print("Histogram of latest cloud_weight message from AMCL particle cloud: ")
    print(bin_heights)

    #Pass through Major test:

kidnapping_threshold_value,x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2
_1,y1_step1_ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean =
loadMajor()

    print("Calling Predict")
    #Call the neural net function
    result =
applyNN([bin_heights],x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2_1,y
1_step1_ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean)
    print("Major Kidnapping NN function produced: ")
    print(result)
    print("Major Kidnapping NN threshold is: ")
    print(kidnapping_threshold_value)

    #Threshold this value and decide if kidnapped.
    if result >= kidnapping_threshold_value:
        print("Major Kidnapping Detected")
            #If kidnapped, call the global relocalization service
            print("Initiating global Localization service.  Redistributing particles across the map.")
            #global_localization()
            kidnap_detected = True
        rospy.logdebug("Major Kidnapping Detected.  NN function produced %f and threshold is
%f",result, kidnapping_threshold_value)


    #Pass through MinorXY test:

kidnapping_threshold_value,x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2
_1,y1_step1_ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean =
loadMinorXY()


    #Call the neural net function
```

```
   result =
applyNN([bin_heights],x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2_1,y
1_step1_ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean)

   print("Minor-(X,Y) Kidnapping NN function produced: ")
   print(result)
   print("Minor-(X,Y) Kidnapping NN threshold is: ")
   print(kidnapping_threshold_value)
   #Threshold this value and decide if kidnapped.
   if result >= kidnapping_threshold_value:
         print("Minor-(X, Y) Kidnapping Detected")
            #If kidnapped, call the global relocalization service
            #global_localization()
            kidnap_detected = True
         rospy.logdebug("Minor XY Kidnapping Detected.  NN function produced %f and
threshold is %f",result, kidnapping_threshold_value)


   #Pass through Minor Theta test:

kidnapping_threshold_value,x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2
_1,y1_step1_ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean =
loadMinorTheta()


   #Call the neural net function
   result =
applyNN([bin_heights],x1_step1_xoffset,x1_step1_gain,x1_step1_ymin,b1,IW1_1,b2,LW2_1,y
1_step1_ymin,y1_step1_gain,y1_step1_xoffset,input_ps_xstd,input_ps_xmean)
   print("Minor-Orientation Kidnapping NN function produced: ")
   print(result)
   print("Minor-Orientation Kidnapping NN threshold is: ")
   print(kidnapping_threshold_value)

   #Threshold this value and decide if kidnapped.
   if result >= kidnapping_threshold_value:
         print("Minor-Orientation Kidnapping Detected")
            #If kidnapped, call the global relocalization service
            #global_localization()
            kidnap_detected = True
         rospy.logdebug("Minor Theta Kidnapping Detected.  NN function produced %f and
threshold is %f",result, kidnapping_threshold_value)

   print("Kidnap Detected flag is set to: " + str(kidnap_detected))
   if(kidnap_detected == True):
            update_sliding_window(True)
```

```
    else:
            update_sliding_window(False)


def listener():

    # in ROS, nodes are unique named. If two nodes with the same
    # node are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'talker' node so that multiple talkers can
    # run simultaenously.

    rospy.init_node('listenerNeuralNetDetect', anonymous=True,log_level=rospy.DEBUG)

    rospy.Subscriber("cloud_weight", filter_covariance_msg, callback)


    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    print("within ROS listener")
    listener()
```

MATLAB Code for Training Neural Network

```
%Neural Network for "No Kidnapping" scenarios:
%Workspace has 55 tables with naming convention:
%    joinedsets[driving method][trial #]

%%Description:
%Read in a data table column-by-column
%Reading from a table returns another table object, so you must convert
%to an array using table2array
%Arrange data into inputs and targets
%Pass to Neural Network

%%Variables:
%x_tw = joinedsetsturtlebotteleop2(:,1);

%column = 3;
%numColumns = 68;
%inputs =[];
%for c = column:numColumns
%    inputs=[inputs table2array([joinedsetsturtlebotteleop2(:,c);
joinedsetsturtlebotteleop3(:,c); joinedsetsturtlebotteleop4(:,c);
joinedsetsturtlebotteleop5(:,c);])];
%end
```

```
%This is what I was testing prior to 1/2/19: [inputs, targets] =
f_LoadNNData('C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\ProlongedDisturbanceNNTrain',',');
%[inputs, targets] = f_LoadNNData('C:\Users\ebrenna8\Documents\Data Files\11-
27-17\NovemberDataCollection\Majortemp',','); %used for major originally
[inputs, targets] = f_LoadNNData('C:\Users\ebrenna8\Documents\Data Files\11-
27-17\NovemberDataCollection\Major\NN-Train',',');
inputs_array = table2array(inputs);
targets_array = table2array(targets);

inputs = inputs_array';

%amcl_gazebo_error = [ joinedsetsturtlebotteleop2(:,'AMCL_GazeboVolume');
joinedsetsturtlebotteleop3(:,'AMCL_GazeboVolume');
joinedsetsturtlebotteleop4(:,'AMCL_GazeboVolume');
joinedsetsturtlebotteleop5(:,'AMCL_GazeboVolume')];
%amcl_gazebo_error = table2array(amcl_gazebo_error);

targets=[targets_array]';

%Configure and run the nn
%This is what I was testing prior to 1/2/19: nn_output =
f_RunNeuralNetwork(inputs, targets, 'C:\Users\ebrenna8\Documents\Data
Files\11-27-
17\NovemberDataCollection\MinorNNtest\MinorTheta75_cloud_weight_hist2.csv');
%nn_output = f_RunNeuralNetwork(inputs, targets,
%'C:\Users\ebrenna8\Documents\Data
%Files\11-27-17\NovemberDataCollection\Majortest\cloud_weight_hist_1.csv');
%used for Major originally
[nn_outputTrain, weightsTrain, biasesTrain,netTrain,input_psTrain] =
f_TrainAndRunNeuralNetwork(inputs, targets, 'C:\Users\ebrenna8\Documents\Data
Files\11-27-17\NovemberDataCollection\RealData\Best Real Trials - 3-23 and 3-
24\Best Real Trials - 3-23 and 3-24\MinorXY1\cloud_weight_hist.csv');
%[input_pn,input_ps,target_pn,target_ps] =
f_NormalizeDatasetsTakenFromTrainAndRunNeuralNetwork(inputs, targets,
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\NN-Test\cloud_weight_hist2.csv');
%[nn_output, weights, biases,net] = f_RunNeuralNetworkWithKnownNet(net,
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\NN-Test\cloud_weight_hist2.csv');
%[nn_output2, weights, biases,net] = f_ReplicateNeuralNetwork(net,
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\NN-Test\cloud_weight_hist2.csv');
%[nn_output2, xweights, xbiases,xnet] =
f_RunNeural_Function('C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\RealData\MinorXY\cloud_weight_hist-
Trial1.csv',input_psTrain,input_psTrain);

function [ nn_output, weights, biases,net,inputs_ps ] =
f_TrainAndRunNeuralNetwork( input_data, target_data, baseFileName )
net = feedforwardnet(15);%Create a net of 10 neurons %5 here with 0.0001 lr
gets it down to 0.04 MSE when just doing teleop
net.trainParam.lr = 0.001; %set learning rate to one over one thousand
net.numLayers = 2;
```

```
%Divide data into 2/3 training and 1/3 validation
net.divideParam.trainRatio = 1/3;
net.divideParam.valRatio = 1/3;
net.divideParam.testRatio = 1/3;

[inputs_std,inputs_ps] = mapstd(input_data);
targets_std = mapstd(target_data);

%Go!
[net,tr] = train(net,inputs_std,targets_std);
weights = net.LW
biases = net.b

%once trained you can use the 'net' object
%
% inputs =[];
% for c = column:numColumns
%     new_inputs=[inputs table2array([joinedsetsturtlebotteleop2(:,c)]);];
% end

%baseFileName = 'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\test\Major9-Test.csv';
%baseFileName = 'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\test\cloud_weight_hist_2.csv';
file_data = readtable(baseFileName,'Delimiter', ',');
%inputs = [file_data(:,'Ratio') file_data(:,'Prev_Ratio')
file_data(:,'Prev_Prev_Ratio')];
inputs = [file_data(:,'B1') file_data(:,'B2') file_data(:,'B3')
file_data(:,'B4') file_data(:,'B5') file_data(:,'B6') file_data(:,'B7')
file_data(:,'B8') file_data(:,'B9') file_data(:,'B10')];
inputs = table2array(inputs);

new_inputs = inputs';

new_inputs_std = mapstd(new_inputs);

nn_output = net(new_inputs_std);


end
```

MATLAB code for plotting AMCL Particle Cloud Data for Visualization

```
function a = FilesToImagesParticleCloud(
InputFileName,SaveFilesToFolder,WeightsInputFileName, GazeboFileName,
AMCLFileName, unique_weights, color_scheme )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

%Clear fiures
clf;

%import data file:
```

```
%InputFileName = 'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\1\particlecloudPreResample.csv';
%SaveFilesToFolder='C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\1\particlecloudpictures\particlecloudpicture
s';
TimesParticleCloudPoints = readtable(InputFileName,'Delimiter', '\t');
TimesParticleCloudPoints = table2array(TimesParticleCloudPoints);

%WeightsInputFileName = 'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\1\cloud_weight.csv';
Weights = readtable(WeightsInputFileName,'Delimiter', '\t');
WeightsArray = table2array(Weights);

GazeboPoses=readtable(GazeboFileName,'Delimiter', '\t');
GazeboPosesArray=table2array(GazeboPoses);

AMCLPoses=readtable(AMCLFileName,'Delimiter', '\t');
AMCLPosesArray=table2array(AMCLPoses);

%unique_weights = sort(unique(WeightsArray(:,3))); %Sort from least to
greatest
%color_scheme = colormap(winter(length(unique_weights))); %create a list of
unique 3-element colors for as many unique particle weights are in the
dataset, arranged lightest-to-darkest  (I think)


d = TimesParticleCloudPoints(:,1);
times = unique(TimesParticleCloudPoints(:,1)); %get distinct time periods
points = [];
counter = 0;
for t=1:size(times)  %for each timestep.

    counter =counter+1;
    points = [];
    for w=1:size(TimesParticleCloudPoints(:,1)) %Go through each row of the
table
        if TimesParticleCloudPoints(w,1) == times(t) %If the current row's
time is the timestep we're dealing with...
            %Record the pose
            AMCL_X = TimesParticleCloudPoints(w,3);
            AMCL_Y = TimesParticleCloudPoints(w,4);
            AMCL_Heading = TimesParticleCloudPoints(w,5);

            %Go to the weights array and find the weight that corresponds
            %to this pose at this timestep

            %WeightsArray(:,1)
            %TimesParticleCloudPoints(w,2)
            %idx =
WeightsArray(WeightsArray(:,1)==TimesParticleCloudPoints(w,1))
            idx = find((WeightsArray(:,1)==TimesParticleCloudPoints(w,1)) &
(WeightsArray(:,2) == TimesParticleCloudPoints(w,2)));
            %This sometimes results in 2, due to duplicate messages in bag
file
           if size(idx,1) > 1
```

```
                idx=idx(size(idx,1));
            end


            if ~isempty(idx)
                AMCL_Weight = WeightsArray(idx,3);
            else
                AMCL_Weight = WeightsArray(1,3);%unique_weights(1);
            end




            %Assign a color

            color_idx = find(unique_weights==AMCL_Weight);
            if ~isempty(color_idx)
                col = color_scheme(color_idx,:);
            else
                %print out the timestep
                col = color_scheme(1,:);
                times(t)

            end


            points = [points; AMCL_X AMCL_Y AMCL_Heading AMCL_Weight col(1,1)
col(1,2) col(1,3)];
        end
    end


    %Plotting x, y, and theta
    [u,v] = pol2cart(points(:,3),0.1); %0.1 %Polar to cartesian (xy)
coordinates. was :,3

    set(gcf,'visible','off') %This prevents a Matlab window w/the figure
     %this lets me plot multiple things with quiver.
    %hold off, fig = quiver(points(:,1), points(:,2), u,v,'color',[0 0 0])
%Black %(Gray=[0.5 0.5 0.5])




    % this makes it suitable for both 2D and 3D
    z =  zeros(length(points),1);
    xy = [points(:,1) points(:,2)];
    xyz=[xy z];
    xyz = num2cell(xyz);
    w = zeros(length(points),1);%angle
    uvw = num2cell([u,v,w]);
```

```
    L=0;
    clf(findobj('type','figure')); %if 'fig' exists,
    %hold on;%Turn on hold within loop so all points get action
    for ii = 1:size(xyz,1)

        L = cellfun(@(x,y) [0;x] + [y;y], ...
            uvw(ii,:), xyz(ii,:),...
            'Uniformoutput', false);
        L = line(L{:});
        set(L, 'color', [points(ii,5) points(ii,6) points(ii,7)]);
        hold on, fig=L;
    end
    hold off; %Turn off hold
    %hold off, fig=L;

 %The below statement plots AMCL particle cloud
 %hold off, fig = quiver(points(:,1), points(:,2), u,v,'color',[0.5 0.5 0.5])
%Black %(Gray=[0.5 0.5 0.5])









    %This part plots the AMCL positoin estimate on top of the point cloud,
    %but something's just not working and arrow looks weird
    %[u,v] = pol2cart(points(1,6),0.1); %0.1
    %hold on, fig = quiver(points(1,4),points(1,5),u,v,'color',[1 0
0],'linewidth',2); %Red
      a_idx=find(AMCLPosesArray(:,1)==times(t));
       %This sometimes results in 2, due to duplicate messages in bag file
      if size(a_idx,1) > 1
               a_idx=a_idx(size(a_idx,1)); %Choose the latest arrival
      end
      if size(a_idx,1)==1
        AMCL_X = AMCLPosesArray(a_idx,2);
        AMCL_Y = AMCLPosesArray(a_idx,3);
        [u,v] = pol2cart(AMCLPosesArray(a_idx,4),0.1); %0.1
        hold on, fig = quiver(AMCL_X,AMCL_Y,u,v,'color',[1 0
0],'linewidth',3); %
        legend('AMCL Pose Estimate');
      end









    %Add a dot for the mean point of thh particle cloud.
    %[u,v] = pol2cart(mean(points(:,3)),0.1); %0.1
    %hold on, fig = quiver(mean(points(:,1)),mean(points(:,2)),u,v,'color',[0
1 0],'linewidth',1); %Green
```

```
     %Add a dot for the mean point of thh particle cloud.
     % for line=1:size(GazeboPoses(:,1))
     %   if GazeboPoses(line,8) == AMCL_X && GazeboPoses(line,9) ==AMCL_Y
     %       Gazebo_X = GazeboPoses(line,2);
     %       Gazebo_Y = GazeboPoses(line,3);
     %       break
     %   end
     % end

     %Add a dot for the Gazebo pose at this timestamp (based on matching AMCL
pose):
     % g_idx=find(GazeboPosesArray(:,1)==times(t));
       %This sometimes results in 2, due to duplicate messages in bag file
     % if size(g_idx,1) > 1
      %         g_idx=g_idx(size(g_idx,1)); %Choose the latest arrival
      %end
      %if size(g_idx,1)==1
      %  Gazebo_X = GazeboPosesArray(g_idx,2);
      %  Gazebo_Y = GazeboPosesArray(g_idx,3);
      %  [u,v] = pol2cart(GazeboPosesArray(g_idx,4),0.1); %0.1
      %  hold on, fig = quiver(Gazebo_X,Gazebo_Y,u,v,'color',[0 0
0],'linewidth',3); %
      %  legend('Gazebo Robot Pose');
      %end



    %legend('Particle Cloud Poses', 'AMCL Pose Estimate (Most Likely Based on
Particle Cloud)', 'Mean Pose of Particle Cloud Poses', 'Ground-truth Gazebo
Robot Pose');


    %Add a title
%     if (times(t) >= 45 && strfind(InputFileName,'Major') &&
~(strfind(InputFileName, '11') || strfind(InputFileName, '12') ||
strfind(InputFileName, '13')))
%         %If Major kidnapping, and if trial 1-10
%         header = strvcat('Post-Kidnapping Event at 45 sec','T = ',
num2str(times(t)));
%     elseif (times(t) < 45 && strfind(InputFileName,'Major') &&
~(strfind(InputFileName, '11') || strfind(InputFileName, '12') ||
strfind(InputFileName, '13')))
%          header = strvcat('Major Kidnapping Example approaching event at 45
sec','T = ', num2str(times(t)));
%
%     %if it's major kidnapping and trial 11-13
%     elseif (times(t) >= 65 && strfind(InputFileName,'Major') &&
(strfind(InputFileName, '11') || strfind(InputFileName, '12') ||
strfind(InputFileName, '13')))
%         %If Major kidnapping, and if trial 11-13
%         header = strvcat('Post-Kidnapping Event at 65 sec','T = ',
num2str(times(t)));
```

```
%     elseif (times(t) < 65 && strfind(InputFileName,'Major') &&
(strfind(InputFileName, '11') || strfind(InputFileName, '12') ||
strfind(InputFileName, '13')))
%          header = strvcat('Major Kidnapping Example approaching event at 65
sec','T = ', num2str(times(t)));
%     elseif strfind(InputFileName, 'Normal')
%         header = strvcat('Normal AMCL Example','T = ', num2str(times(t)));
%     else
%          header = strvcat('Major Kidnapping Example','T = ',
num2str(times(t)));
%     end

    if (isempty(strfind(InputFileName,'Normal')))
        header= strvcat('Major Kidnapping Example','T = ',
num2str(times(t)));
    else
     header = strvcat('Normal AMCL Example','T = ', num2str(times(t)));
    end


    caxis([ min(unique_weights) max(unique_weights)]);
    colorbar;


    title(header);
    axis([-3 3 -3 3]);
    saveas(fig, strcat(SaveFilesToFolder, int2str(counter),'.png'));
    clf(fig,'reset');
end




end
```

Python Script for Converting ROS Bag File into Individual CSVs for Each ROS Message Topic

```
import rosbag
import os
import sys
import math
import tf

from tf.transformations import euler_from_quaternion
import matplotlib.pyplot as pl
import numpy as np
```

```
def get_amcl_pose_covariance(f, out_file, topic_name):

	out_file.write('Time\tCovariance0\tCovariance1\tCovariance2\tCovariance3\tCovariance
4\tCovarianace5\tCovariance6\n')
	for topic, msg,t in f.read_messages(topics = [topic_name]):

		out_file.write(str(int(str(t))/1000000000) + "." + str(int(str(t))%1000000000) +
"\t")
		for element in msg.pose.covariance:
			out_file.write(str(element) + '\t')
		out_file.write('\n')



def get_array(f, out_file, topic_name):
	out_file.write('Time\tParticleNumber\tParticleWeight\n')

	for topic, msg, t in f.read_messages(topics=[topic_name]):
		counter=0
		for element in msg.cov:
			#Timestamp
			out_file.write(str(int(str(t))/1000000000) + "." +
str(int(str(t))%1000000000) + "\t")
			out_file.write(str(counter) + "\t" + str(element) + "\n")
			counter=counter+1


def get_cloud_weight_hist_ratio(f, out_file, topic_name):
	out_file.write('Time\tRatio\tNumBins\n')

	for topic, msg, t in f.read_messages(topics=[topic_name]):
		counter=0
		data_list = []
		for element in msg.cov:
			data_list.append(element)

		bin_heights, bin_boundaries, patches = pl.hist(np.asarray(data_list), 10)

		sorted_array = sorted(bin_heights, reverse=True) #Reverse True is important -
want tallest to shortest
		ratio = sorted_array[0]/sorted_array[1] #height of tallest histogram bin to 2nd
tallest

		#Timestamp
		out_file.write(str(int(str(t))/1000000000) + "." + str(int(str(t))%1000000000) +
"\t")
```

```
                out_file.write(str(ratio) + "\t" + str(10) + "\n")


def get_cloud_weight_hist(f, out_file, topic_name, kidnapping):
        out_file.write('Time,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,Kidnapped\n')

        for topic, msg, t in f.read_messages(topics=[topic_name]):
                counter=0
                data_list = []
                for element in msg.cov:
                        data_list.append(element)

                bin_heights, bin_boundaries, patches = pl.hist(np.asarray(data_list), 10)

                #Timestamp
                out_file.write(str(int(str(t))/1000000000) + "." + str(int(str(t))%1000000000) +
","")
                for x in bin_heights:
                        out_file.write(str(x) + ',')

                #Add Kidnapping indicator of 0. Will edit rest manually.
                if kidnapping==False:
                        out_file.write(str(0) + ",")
                else:
                        if ((int(str(t))/1000000000) < 45): #45 is general halfwaypoint. Requires
manual editing.
                                out_file.write(str(0))
                        else:
                                out_file.write(str(1))

                out_file.write("\n") #Finish with a new line char



def get_topic_pose(f, out_file, topic_name):

        out_file.write('Time\tX(meters)\tY(meters)\tHeading(Radians)\n')
        for topic, msg,t in f.read_messages(topics = [topic_name]):

                out_file.write(str(int(str(t))/1000000000) + "." + str(int(str(t))%1000000000) +
"\t")
                x = msg.pose.pose.position.x
                y = msg.pose.pose.position.y
                quat = (
                        msg.pose.pose.orientation.x,
```

```
                        msg.pose.pose.orientation.y,
                        msg.pose.pose.orientation.z,
                        msg.pose.pose.orientation.w)
                yaw = tf.transformations.euler_from_quaternion(quat)[2]
                out_file.write(str(x) + "\t" + str(y) + '\t' + str(yaw) + "\n")


def get_rosout_amcl_messages(f, out_file, topic_name):
        out_file.write('Time\tMessage\n')
        for topic, msg,t in f.read_messages(topics = [topic_name]):

                out_file.write(str(int(str(t))/1000000000) + "." + str(int(str(t))%1000000000) +
"\t" + str(msg.msg) + "\n")




def get_particle_cloud(f, out_file, topic_name):
        out_file.write('Time\tElement\tX(meters)\tY(meters)\tHeading(Radians)' + '\n')
        #for topic, msg, t in f.read_messages(topics = ['/particlecloud']):

        for topic, msg, t in f.read_messages(topics = [topic_name]):
                counter = 0
                particle_cloud_seq= msg.header.seq

                #get the associated value of the amcl_pose - i.e., the position estimate according
to AMCL


                for element in msg.poses:
                        #Timestamp:
                        out_file.write(str(int(str(t))/1000000000) + "." +
str(int(str(t))%1000000000) + "\t")

                        x=element.position.x
                        y=element.position.y
                        qx =element.orientation.x
                        qy = element.orientation.y
                        qz = element.orientation.z
                        qw = element.orientation.w


                        #Dumb quaternion math:
                        #XX = qx * qx
                        #XZ = qx * qz
```

```
                        #XW = qx * qw
                        #YY = qy * qy
                        #YW = qy * qw
                        #YZ = qy * qz
                        #ZZ = qz * qz
                        #yaw = math.atan2(2 * YW - 2 * XZ, 1-2*YY-2*ZZ)
                        quaternion = (
                           qx,
                           qy,
                           qz,
                           qw)

                        euler = tf.transformations.euler_from_quaternion(quaternion)
                        #print euler[2]
                        out_file.write(str(counter) + "\t" + str(x) + "\t" + str(y) + "\t" +
str(euler[2]) + "\n") # + "\t" + str(qx) + "\t" + str(qy) + "\t" + str(qz) + "\t" + str(qw) + "\t" +
str(amcl_x) + "\t" + str(amcl_y) + "\t" + str(amcl_yaw) + "\n")
                        counter = counter + 1 #increment counter

        return


if __name__=="__main__":
        if len(sys.argv) == 3:
                print 'Enough arguments were supplied'
        else:
                print 'NOT ENOUGH ARGUMENTS SUPPLIED. NEED BAG NAME AND
OUTPUT FILE DIRECTORY'
                sys.exit()


        bag_file =sys.argv[1]
        print 'Opening: '
        print bag_file
        out_file_path = sys.argv[2]
        print 'Saving to...'
        print out_file_path

        f=rosbag.Bag(bag_file)
        #topics = f.get_type_and_topic_info()[1].keys()
        #print 'Topics: \n'
        #print topics
        #for i in range(0, len(f.get_type_and_topic_info()[1].values())):
        #       print f.get_type_and_topic_info()[1].values()[i][0]
```

```
#The idea here is to go through each of the important topics and print them out to their own CSV

#Particle Cloud - post-resample
outFile = open(out_file_path+'particlecloud.csv', 'w')
print 'Path:'
print outFile
get_particle_cloud(f, outFile, '/particlecloud')
outFile.close()



#Particle Cloud - pre-resample
outFile = open(out_file_path+'particlecloudPreResample.csv', 'w')
print 'Path:'
print outFile
get_particle_cloud(f, outFile, '/particlecloudPreResample')
outFile.close()


#AMCL
outFile = open(out_file_path+'amcl_pose.csv', 'w')
print 'Path:'
print outFile
get_topic_pose(f, outFile, '/amcl_pose')
outFile.close()


#Gazebo Pose
outFile = open(out_file_path+'gazebo_robot_pose.csv', 'w')
print 'Path:'
print outFile
get_topic_pose(f, outFile, '/gazebo_robot_pose')
outFile.close()


#Cloud Weight
outFile = open(out_file_path+'cloud_weight.csv', 'w')
print 'Path:'
print outFile
get_array(f, outFile, '/cloud_weight')


#Histogram ratio of cloud weights
outFile = open(out_file_path+'cloud_weight_hist.csv', 'w')
print 'Path:'
```

```
print outFile
#for assigning 0's and 1's
if 'Normal' in out_file_path:
        kidnapping=False
else:
        kidnapping=True
get_cloud_weight_hist(f, outFile, '/cloud_weight', kidnapping)


#AMCL Covariancee
outFile = open(out_file_path+'amcl_covariance.csv', 'w')
print 'Path:'
print outFile
get_amcl_pose_covariance(f, outFile, '/amcl_pose')

#Rosout AMCL messages
outFile = open(out_file_path+'rosout_amcl_messages.csv', 'w')
print 'Path:'
print outFile
get_rosout_amcl_messages(f, outFile, '/rosout')

print 'Finished'
```

R Code for Joining AMCL and Gazebo Datasets

```
#1
#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Major\\Test321\\Test321-
1\\1\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Major\\Test321\\Test321-
1\\1\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
```

```
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Major\\Test321\\Test321-1\\1\\amcl_gazebo_poses_joined.csv",
x=final)

#2

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\2\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\2\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
```

```
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\2\\amcl_gazebo_poses_joined.csv", x=final)

#3

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\3\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\3\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\3\\amcl_gazebo_poses_joined.csv", x=final)

#4

#This script joins the Gazebo data to each timestep of the AMCL dataset
```

```
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\4\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\4\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\4\\amcl_gazebo_poses_joined.csv", x=final)

#5

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\5\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\5\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
```

```
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\5\\amcl_gazebo_poses_joined.csv", x=final)

#6

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\6\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\6\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
```

```
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\6\\amcl_gazebo_poses_joined.csv", x=final)
```

#7

```
#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\7\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\7\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
```

```
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\7\\amcl_gazebo_poses_joined.csv", x=final)

#8
#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\8\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\8\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\8\\amcl_gazebo_poses_joined.csv", x=final)

#9

#This script joins the Gazebo data to each timestep of the AMCL dataset
```

```
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\9\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\9\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\9\\amcl_gazebo_poses_joined.csv", x=final)

#10


#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\10\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\10\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
```

```
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\10\\amcl_gazebo_poses_joined.csv", x=final)

#1

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\11\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\11\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep
```

```
#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)

#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\11\\amcl_gazebo_poses_joined.csv", x=final)

#2

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\12\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\12\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
```

```
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y - final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)

#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-17\\NovemberDataCollection\\Minor\\XY 0.1 Only-Raw\\75s\\12\\amcl_gazebo_poses_joined.csv", x=final)

#3

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-17\\NovemberDataCollection\\Minor\\XY 0.1 Only-Raw\\75s\\13\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-17\\NovemberDataCollection\\Minor\\XY 0.1 Only-Raw\\75s\\13\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y - final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
```

```
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\13\\amcl_gazebo_poses_joined.csv", x=final)

#4

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\14\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\14\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\14\\amcl_gazebo_poses_joined.csv", x=final)

#5

#This script joins the Gazebo data to each timestep of the AMCL dataset
```

```
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\15\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\15\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\15\\amcl_gazebo_poses_joined.csv", x=final)

#6

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\16\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\16\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
```

```
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)

#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\16\\amcl_gazebo_poses_joined.csv", x=final)

#7

#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\17\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\17\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep
```

```
#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)
#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\17\\amcl_gazebo_poses_joined.csv", x=final)
```

#8

```
#This script joins the Gazebo data to each timestep of the AMCL dataset
gazebo<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\18\\gazebo_robot_pose.csv",header=TRUE,sep="\t")
amcl<-read.csv("C:\\Users\\ebrenna8\\Documents\\Data Files\\11-27-
17\\NovemberDataCollection\\Minor\\XY 0.1 Only-
Raw\\75s\\18\\amcl_pose.csv",header=TRUE,sep="\t")
#Rename AMCL columns so the datasets have different var names. Merge wasn't working when
tables had same col names
colnames(amcl)[colnames(amcl)=="X.meters."] <- "AMCL_X"
colnames(amcl)[colnames(amcl)=="Y.meters."] <- "AMCL_Y"
colnames(amcl)[colnames(amcl)=="Heading.Radians."] <- "AMCL_Heading"
colnames(gazebo)[colnames(gazebo)=="X.meters."] <- "Gazebo_X"
colnames(gazebo)[colnames(gazebo)=="Y.meters."] <- "Gazebo_Y"
colnames(gazebo)[colnames(gazebo)=="Heading.Radians."] <- "Gazebo_Heading"

myfulldata<-merge(amcl,gazebo)      #This grabbed all the Gazebo data for each AMCL timestep

#Grabbing the first one for each timestep:
ordered_data<-myfulldata[order(myfulldata$Time),] #sort
final<-ordered_data[!duplicated(ordered_data$Time),] #grab first
#Compare AMCL to Gazebo in each dimension
final$Delta_X<-abs(final$AMCL_X - final$Gazebo_X)
final$Delta_Y<-abs(final$AMCL_Y - final$Gazebo_Y)
final$Delta_Heading<-abs(final$AMCL_Heading - final$Gazebo_Heading)
final$XYDistFormula<-sqrt((final$AMCL_X - final$Gazebo_X)**2 + (final$AMCL_Y -
final$Gazebo_Y)**2)
```

#Multiply the differences, do a log so that the numbers aren't so tiny
final$Log_Of_Deltas_Multiplied = log(final$Delta_X * final$Delta_Y * final$Delta_Heading)
final$Log_Of_XYDist_x_Delta_Heading<-log(final$XYDistFormula * final$Delta_Heading)

#How to savea a CSV:
write.csv(file="C:\\users\\ebrenna8\\Documents\\Data Files\\11-27-17\\NovemberDataCollection\\Minor\\XY 0.1 Only-Raw\\75s\\18\\amcl_gazebo_poses_joined.csv", x=final)

MATLAB Experiment with Representing Particle Filter Data as a 3D Mesh

```
function a = FilesToVolume(
InputFileName,SaveFilesToFolder,SaveFilesToFolderDots,WeightsInputFileName,
GazeboFileName, unique_weights, AllX,AllY)
%Clear fiures
clf;

%import data file:
%InputFileName = 'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\1\particlecloudPreResample.csv';
%SaveFilesToFolder='C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\1\particlecloudpictures\particlecloudpicture
s';
TimesParticleCloudPoints = readtable(InputFileName,'Delimiter', '\t');
TimesParticleCloudPoints = table2array(TimesParticleCloudPoints);

%WeightsInputFileName = 'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\1\cloud_weight.csv';
Weights = readtable(WeightsInputFileName,'Delimiter', '\t');
WeightsArray = table2array(Weights);

GazeboAMCLData=readtable(GazeboFileName,'Delimiter', ',');
%GazeboAMCLArray = table2array(GazeboAMCLData);
%unique_weights = sort(unique(WeightsArray(:,3))); %Sort from least to
greatest
%color_scheme = colormap(winter(length(unique_weights))); %create a list of
unique 3-element colors for as many unique particle weights are in the
dataset, arranged lightest-to-darkest  (I think)


d = TimesParticleCloudPoints(:,1);
times = unique(TimesParticleCloudPoints(:,1)); %get distinct time periods
points = [];
counter = 0;
for t=1:size(times)  %for each timestep.

    counter =counter+1;
    points = [];
    for w=1:size(TimesParticleCloudPoints(:,1)) %Go through each row of the
table
        if TimesParticleCloudPoints(w,1) == times(t) %If the current row's
time is the timestep we're dealing with...
            %Record the pose
```

```
            AMCL_X = TimesParticleCloudPoints(w,3);
            AMCL_Y = TimesParticleCloudPoints(w,4);
            AMCL_Heading = TimesParticleCloudPoints(w,5);

            %Go to the weights array and find the weight that corresponds
            %to this pose at this timestep

            %WeightsArray(:,1)
            %TimesParticleCloudPoints(w,2)
            %idx =
WeightsArray(WeightsArray(:,1)==TimesParticleCloudPoints(w,1))
            idx = find((WeightsArray(:,1)==TimesParticleCloudPoints(w,1)) &
(WeightsArray(:,2) == TimesParticleCloudPoints(w,2)));
            %This sometimes results in 2, due to duplicate messages in bag
file
           if size(idx,1) > 1
               idx=idx(size(idx,1));
           end


            if ~isempty(idx)
                AMCL_Weight = WeightsArray(idx,3);
            else
                AMCL_Weight = WeightsArray(1,3);%unique_weights(1);
            end






            points = [points; AMCL_X AMCL_Y AMCL_Heading AMCL_Weight];
        end
    end

   x = points(:,1);
   y=points(:,2);
   z=points(:,3);
   weights=points(:,4);


   min(x)
   max(x)
   min(y)
   max(y)
   x = x-min(x);%normalize at 0
   y=y-min(y);
   f = fit([x, y], weights, 'linearinterp');
%q2 = quad2d(f,211.1,211.4,-48.35,-48,'AbsTol',0.01)
q2 = quad2d(f, 0,max(x),0,max(y),'AbsTol',0.01)%,211.1,211.4,-48.35,-
48,'AbsTol',0.01)
%q2 = integral3(f,
0,max(x),0,max(y),0,max(weights),'AbsTol',0.01)%,211.1,211.4,-48.35,-
48,'AbsTol',0.01)
```

```
F2 = scatteredInterpolant(x,y,weights);



 %figure;
%[xi,yi] = meshgrid(min(AllX):max(AllX), min(AllY):max(AllY));%doesn't work
[xi,yi] = meshgrid(min(x):0.01:max(x), min(y):0.01:max(y));%0.01 allows
colors, i guess
zi = griddata(x,y,weights,xi,yi);
fig=surf(xi,yi,zi);
xlabel('X (meters)');
ylabel('Y (meters)');
zlabel('Particle Weight');
%axis([-10 10 -10 10 -6.28 6.28]);

if (isempty(strfind(InputFileName,'Normal')))
        header= strvcat('Major Kidnapping Example','T = ',
num2str(times(t)));
    else
     header = strvcat('Normal AMCL Example','T = ', num2str(times(t)));
end


 g_idx=find(GazeboAMCLArray(:,2)==times(t));
       %This sometimes results in 2, due to duplicate messages in bag file
 if size(g_idx,1) > 1
               g_idx=g_idx(size(g_idx,1)); %Choose the latest arrival
      end
      if size(g_idx,1)==1
          header=strvcat(header,num2str(GazeboAMCLArray(g_idx,9)));

      end




title(header);
    %axis([-3 3 -3 3]);
    saveas(fig, strcat(SaveFilesToFolder, int2str(counter),'.png'));
    clf(fig,'reset');

%Dots

long = x;
lat = y;
rural = z;
fatalities = points(:,4);%hwydata(:,11);                          % fatalities
data

fig =scatter3(long,lat,rural,40,fatalities);    % draw the scatter plot
%hold on, fig = scatter3(-9,-9,-4,.5,'filled', 'MarkerFaceColor','red');
%%Not working
```

```
%ax = gca;
%ax.XDir = 'reverse';
%view(-31,14)
xlabel('X (meters)')
ylabel('Y (meters)')
zlabel('Z (Heading)')

cb = colorbar;                                          % create and label the
colorbar
cb.Label.String = 'Particle weight';
caxis([ min(unique_weights) max(unique_weights)]);


%axis([min(AllX) max(AllX) min(AllY) max(AllY) -6.28 6.28]);
axis([-10 10 -10 10 -6.28 6.28]);
if (isempty(strfind(InputFileName,'Normal')))
        header= strvcat('Major Kidnapping Example','T = ',
num2str(times(t)));
    else
     header = strvcat('Normal AMCL Example','T = ', num2str(times(t)));
   end
    %caxis([ min(unique_weights) max(unique_weights)]);
    %colorbar;


    title(header);
    %axis([-3 3 -3 3]);
    saveas(fig, strcat(SaveFilesToFolderDots, int2str(counter),'.png'));
    clf(fig,'reset');



end
```

MATLAB Script for Creating a Standard Color Scheme for Particle Cloud Data

```
%calculate colorscheme to use for all plots based on
%max, min particle weights we've ever seen.

FileNames = {

'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\1\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\2\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\3\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\4\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\5\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\6\cloud_weight.csv'
```

```
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\7\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\8\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\9\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Normal\10\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\1\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\2\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\3\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\4\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\5\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\6\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\7\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\8\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\9\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\10\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\11\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\12\cloud_weight.csv'
'C:\Users\ebrenna8\Documents\Data Files\11-27-
17\NovemberDataCollection\Major\13\cloud_weight.csv'
}

WeightsArrayAll=[];

for fn=1:length(FileNames)
%Read-in a file
WeightsInputFileName = FileNames{fn}; %sprintf(FileNames{fn});

Weights = readtable(WeightsInputFileName,'Delimiter', '\t');
WeightsArray = table2array(Weights);

WeightsArrayAll = [WeightsArrayAll;WeightsArray]; %Append to list
end


%load the weights and use colormap
unique_weights = sort(unique(WeightsArrayAll(:,3))); %Sort from least to
greatest
color_scheme = colormap(jet(length(unique_weights))); %create a list of
unique 3-element colors for as many unique particle weights are in the
dataset, arranged lightest-to-darkest  (I think)
```

Appendix M

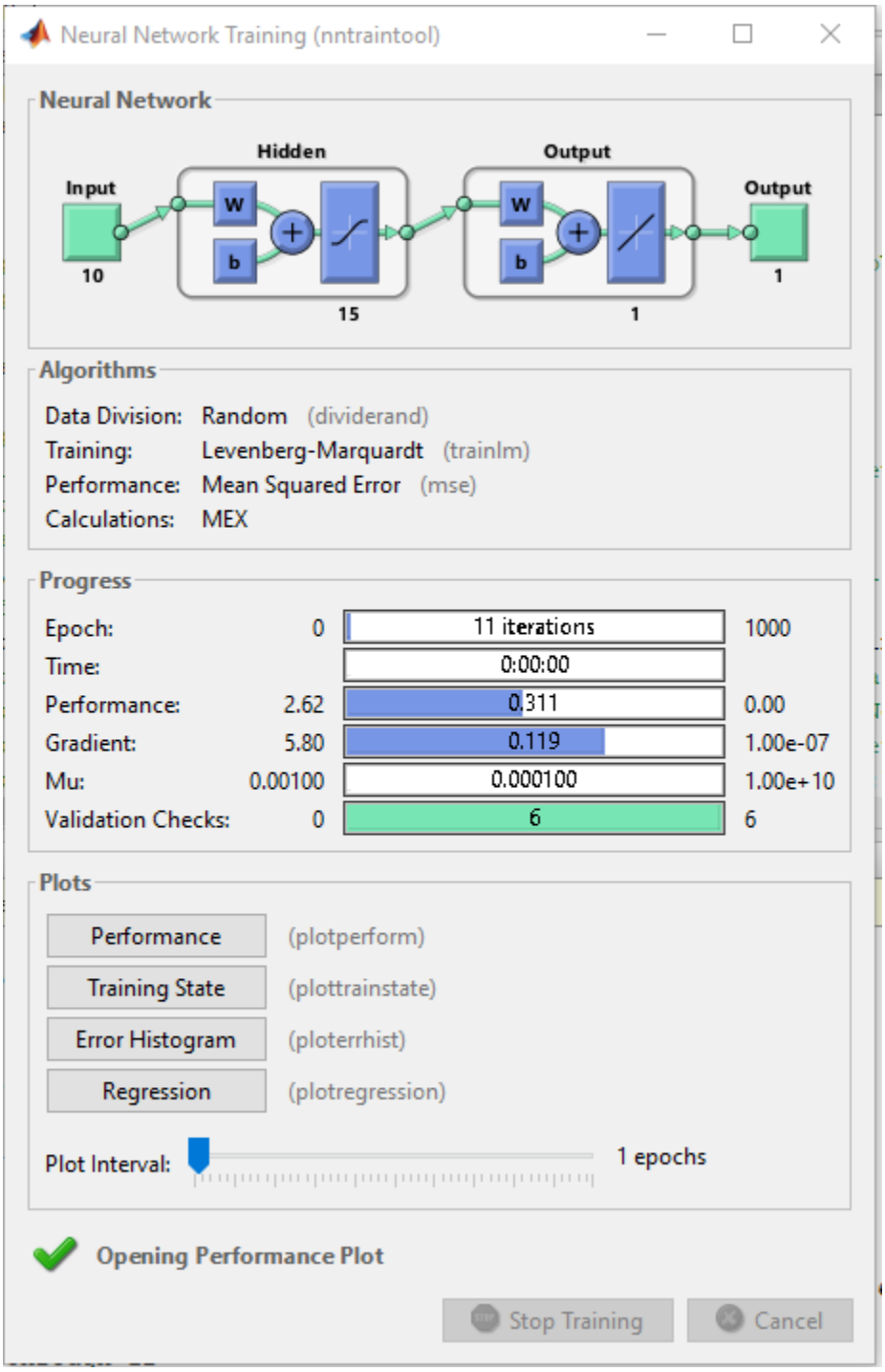


Figure M1: Neural Network Training Results

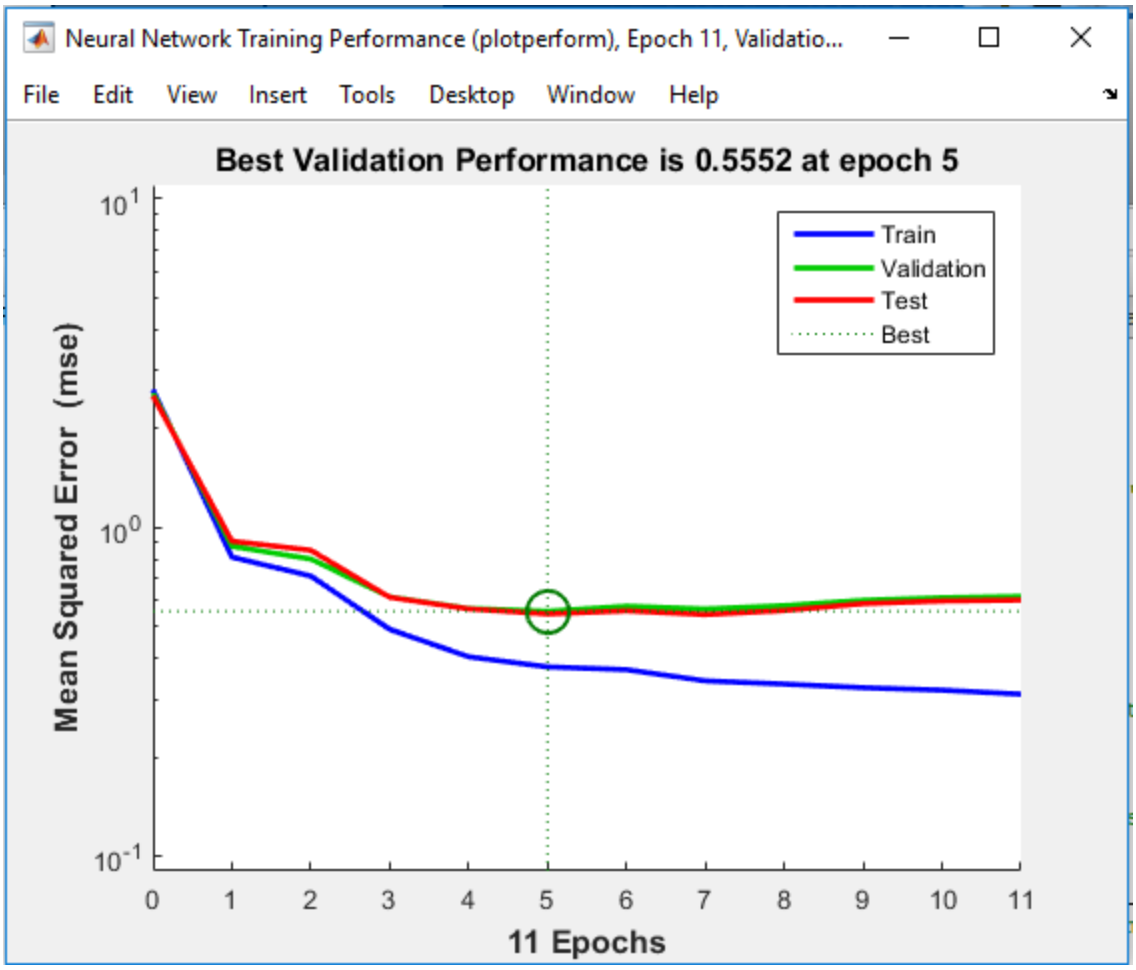


Figure M2:Performance

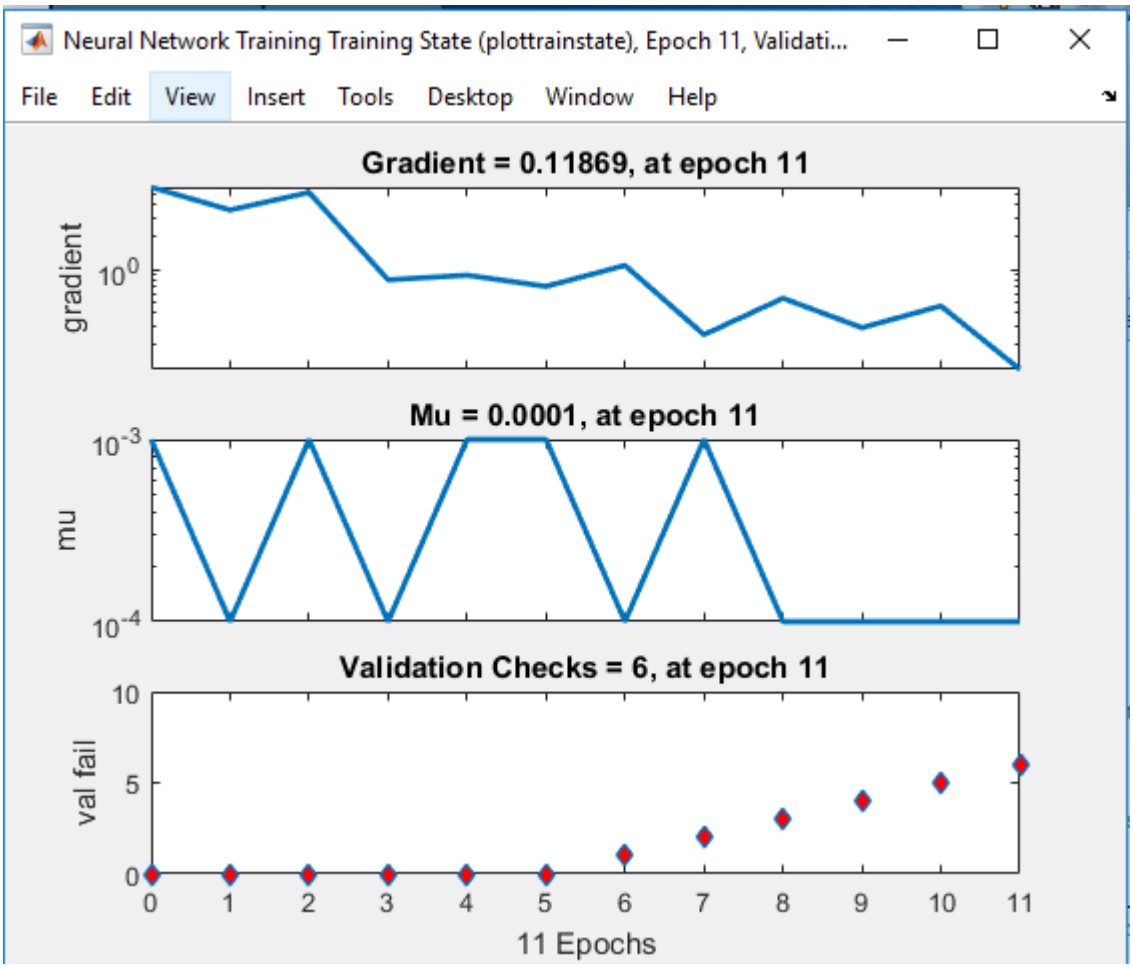


Figure M3: Training State

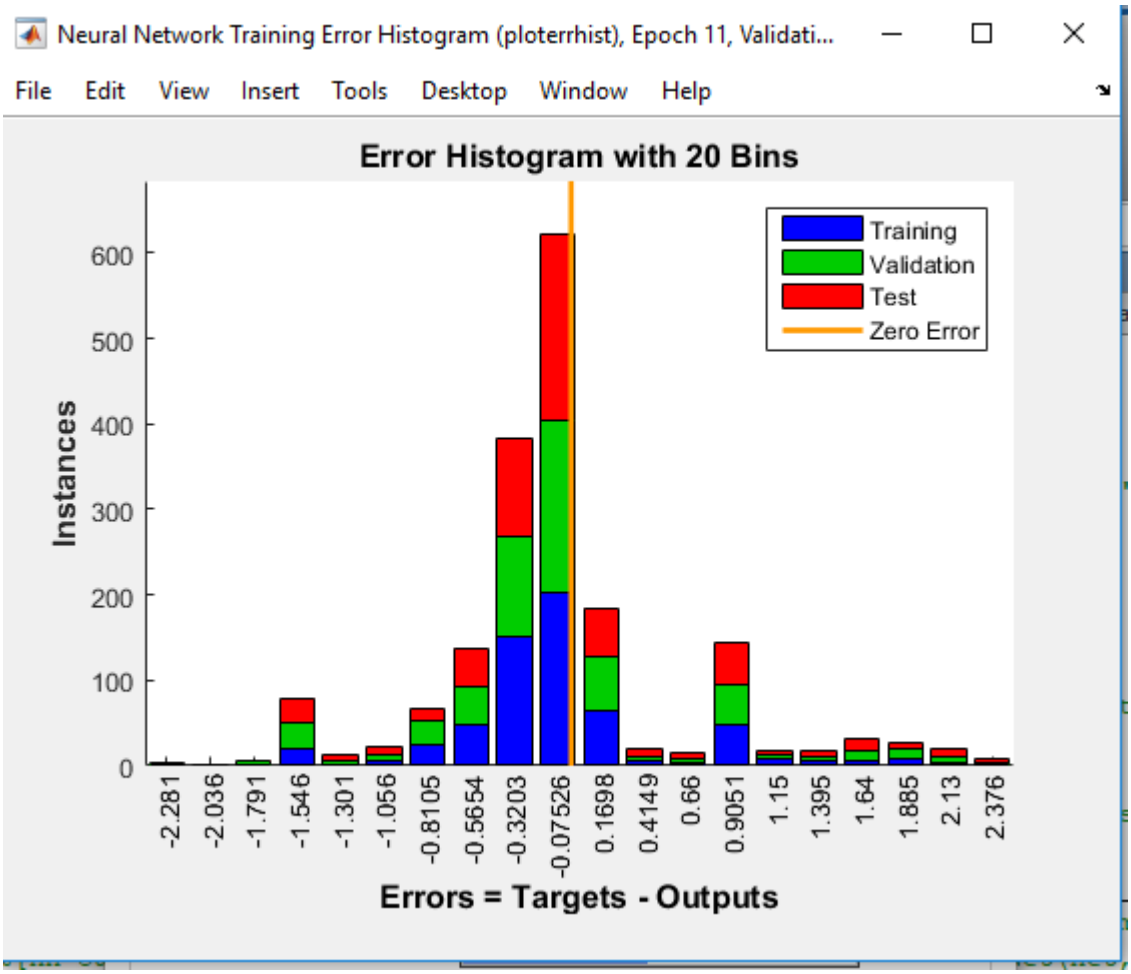


Figure M436: Error Histogram

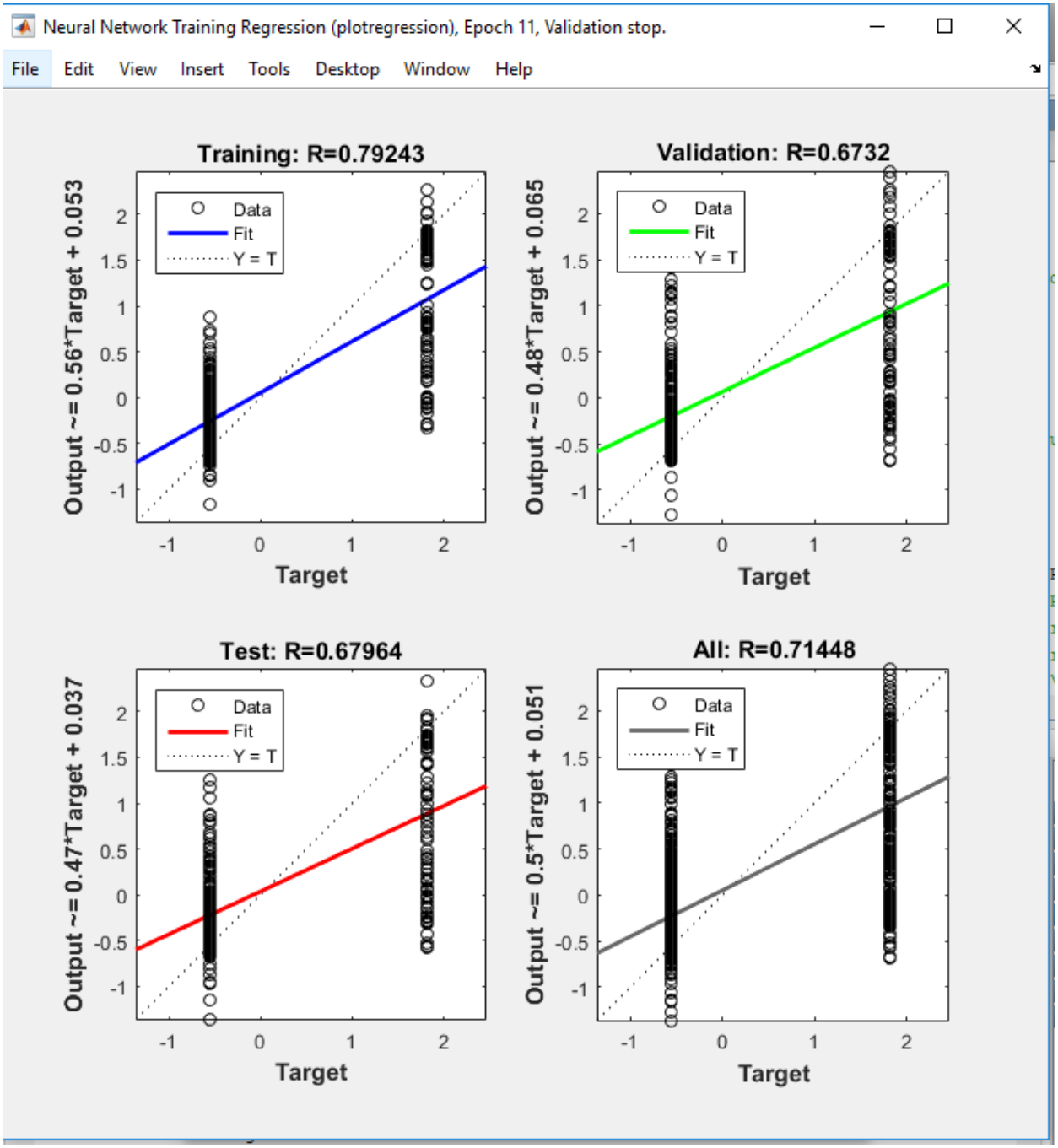


Figure M5: Regression

BIBLIOGRAPHY

[1] S. Lee, S. Lee, and S. Baek, "Vision-Based Kidnap Recovery with SLAM for Home Cleaning Robots," *Journal of Intelligent & Robotic Systems,* vol. 67, pp. 7-24, July 2012.

[2] M. Simon, "This Incredible Hospital Robot Is Saving Lives. Also, I Hate It," *Wired,* p.*1*, February 2, 2015.

[3] R. Li et al., "Spirit Rover Localization and Topographic Mapping at the Landing Site of Gusev Crater, Mars," *Journal of Geophysical Research,* vol. III, 2006.

[4] C. C. Ward and K. Iagnemma, "A Dynamic-Model-Based Wheel Slip Detector for Mobile Robots on Outdoor Terrain," in IEEE Transactions on Robotics, vol. 24, no. 4, pp. 821-831, Aug. 2008.

[5] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics. Cambridge: MIT Press, 2006.

[6] R. Siegward, I. R. Nourbakhsh, D. Scaramuzza, Introduction to Autonomous Mobile Robots. Cambridge: MIT Press, 2011.

[7] R. Guyonneau, S. Lagrange, L. Hardouin, and P. Lucidarme. "The Kidnapping Problem of Mobile Robots: A Set Membership Approach," in *7th National Conference on "Control Architectures of Robots," Nancy, France, May 10-11, 2012.*

[8] D. Fox,W.Burgard, F. Dellaert and S. Thrun. "Monte Carlo localization: efficient position estimation for mobile robots*," Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence (AAAI '99/IAAI '99)*. American Association for Artificial Intelligence, Orlando, Florida, USA, pp. 343-349, July 18-22, 1999.

[9] S. Koenig, J. Mitchell, A. Mudgal, and C. Tovey, "A Near-Tight Approximation Algorithm for the Robot Localization Problem," SIAM Journal on Computing, vol. 39 Issue 2, pp. 461-490, June 2009.

[10] Y. Tian and S. Ma, "A double guarantee kidnapping detection in simultaneous localization and mapping," 2014 International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI), Beijing, 2014, pp. 1-6.

[11] D. Campbell and M. Whitty, "Metric-Based Detection of Robot Kidnapping," in *2013 European Conference on Mobile Robots (ECMR), Barcelona, Spain, September 25-27, 2013*, IEEE, 2013. pp 192-97.

[12] I. Bukhori and Z. H. Ismail, "Detection of kidnapped robot problem in Monte Carlo localization based on the natural displacement of the robot," *International Journal of Advanced Robotic Systems*, pp. 1-6, July-August 2017.

[13] L. Zhang, R. Zapata and P. Lepinay, "Self-Adaptive Monte Carlo for Single-Robot and Multi-Robot Localization," 2009 IEEE International Conference on Automation and Logistics, Shenyang, 2009, pp. 1927-1933.

[14] Z. Duan, Z. Cai, and H. Min. "Robust dead reckoning system for mobile robots based on particle filter and raw range scan.," *Sensors (Basel)*, vol. 14 issue 9, pp. 16532-62, Sept. 2014.

[15] Ozkil A, Fan Z, Xiao J, Dawids S, Kristensen J, Christensen K. Mapping of multi-floor buildings: a barometric approach. In: Proceedings on IEEE/RSJ international conference on intelligent robots and systems, San Francisco, USA; 2011. p. 847–52.

[16] C. Yi and B. Choi, "Detection and Recovery for Kidnapped-Robot Problem Using Measurement Entropy," *International Conference on Grid and Distributed Computing, Communications in Computer and Information Science*, vol 261, pp. 293-299, 2011.

[17] Y. Kim and H. Bang, "Vision-based navigation for unmanned aircraft using ground feature points and terrain elevation data," *Proceedings of the Institution of Mechanical Engineers, Part : Journal of Aerospace Engineering*, vol. 232 issue 7, pp. 1334-1346, June 2018.
[18] W. Qian et al., "Manipulation task simulation using ROS and Gazebo," 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014), Bali, 2014, pp. 2594-2598.
[19] Gazebosim.org, "Tutorial: ROS Plugin," Open Source Robotics Foundation, 2014. [Online]. Available: *http://gazebosim.org/tutorials?tut=ros_plugins*. [Accessed Oct. 13, 2018].
[20] ROS.org, "Publishing Odometry Information over ROS," ROS.org. [Online]. Available: *http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom*
[21] B. Gerkey and K. Stoy, "amcl", GNU Lesser General Public License, July 31, 2017. [Online]. Available: *https://github.com/ros-planning/navigation/tree/melodic-devel/amcl*. [Accessed Oct. 7, 2018].
[22] M. Wise *et al*., "ROS Wiki: turtlebot_navigation," ROS.org, Feb. 2011. [Online]. Available: *http://wiki.ros.org/turtlebot_navigation*. [Accessed Oct. 13, 2018].
[23] W. Woodall *et al*., "ROS Wiki: Overlaying with catkin workspaces," ROS.org, April 2018. [Online] Available: *http://wiki.ros.org/catkin/Tutorials/workspace_overlaying*. [Accessed Oct. 13, 2018].
[24] K. Corley *et al*., "ROS Wiki: Defining Custom Messages," ROS.org, April 2018. [Online] Available: *http://wiki.ros.org/ROS/Tutorials/DefiningCustomMessages*. [Accessed Oct. 13, 2018].
[25] ROS Answers, "AMCL Particle Weights Are Always Equal – Why?," answers.ros.org. [Online]. Available: *https://answers.ros.org/question/275574/amcl-particle-weights-are-always-equal-why/*
[26] Willow Garage, "turtlebot_simulator," turtlebot, Sept. 18, 2017. [Online]. Available: *https://github.com/turtlebot/turtlebot_simulator/tree/indigo/turtlebot_gazebo/launch*. [Accessed Oct. 13, 2018].
[27] M. Wise *et al*., "ROS Wiki: turtlebot_gazebo," ROS.org, April 2018. [Online]. Available: *http://wiki.ros.org/turtlebot_gazebo*. [Accessed Oct. 13, 2018].
[28] Gazebo. "Timer Class Reference," Open Source Robotics Foundation, 2014. [Online]. Available: *http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1common_1_1Timer.html*. [Accessed Oct. 13, 2018].
[29] Gazebosim.org, "Topics Subscription," Open Source Robotics Foundation, 2014. [Online]. Available: *http://gazebosim.org/tutorials?tut=topics_subscribed&ver=1.9*. [Accessed Oct. 13, 2018].
[30] gazebosim.org, "Gazebo API Gazebo Transport," gazebosim.org. [Online]. Available: *http://gazebosim.org/api/dev/group__gazebo__transport.html#details*
[31] P. *Mihelich et al*., "rosbag," ROS.org, Aug. 2009. [Online]. Available: *http://wiki.ros.org/rosbag*. [Accessed Oct. 13, 2018].
[38] aktaylor08, "RosbagPandas," github.com. [Online]. Available: *https://github.com/aktaylor08/RosbagPandas*
[39] Answers.ros.org, "Cannot load message class for … Are your message built?," ROS Answers, 2014. [Online]. Available: *https://answers.ros.org/question/187071/cannot-load-message-class-for-are-your-messages-built/*

[40] M. Wise *et al*., "ROS Wiki: gmapping," ROS.org, July 2009. [Online]. Available: *http://wiki.ros.org/gmapping*. [Accessed Oct. 13, 2018].
[41] Answers.ros.org, "Orienting AMCL Map/Global_Costmap," ROS Answers, 2017. [Online]. Available: *https://answers.ros.org/question/264397/orienting-amcl-mapglobal_costmap/*
[42] Github.com, "Static Layer Ignores Map Orientation #166," ros-planning, 2014. [Online]. Available: *https://github.com/ros-planning/navigation/issues/166*
[43] Github.com, "AMCL ignores map orientation #217," ros-planning, 2014. [Online]. Available: *https://github.com/ros-planning/navigation/issues/217*
[44] Github.com, "costmap2d does not respect yaw in map origin parameter #661.," ros-planning, 2018. [Online]. Available: *https://github.com/ros-planning/navigation/issues/661*
[45] P. Allen, "CS W4733 NOTES - Differential Drive Robots," 2015. [Online]. Available: http://www.cs.columbia.edu/~allen/F15/NOTES/icckinematics.pdf
[46] E. Brennan, "ThesisResearch-KidnappedRobotProblem," 2019. [Online]. Available: https://github.com/neptune1492/ThesisResearch-KidnappedRobotProblem
[47] discourse.ros.org, "ROS Discourse," 2019. [Online]. Available: https://discourse.ros.org

Vita Auctoris

Elizabeth Brennan received a Bachelor's of Science degree in Computer Science and Applied Mathematics as a double-major with a concentration in Accounting from Fontbonne University in Saint Louis in 2014. While at Saint Louis University, she was a Teaching Assistant for an undergraduate Network Programming course and completed a Certificate in University Teaching Skills from the Paul C. Reinert, S.J. Center for Transformative Teaching and Learning. In 2017, she was awarded First Place in the "Paper – Physical Sciences" category at the Graduate Student Symposium for her presentation on "The Kidnapped Robot Problem." Employed as a full-time software developer in Saint Louis while finishing this thesis research, she developed a full-text search application.