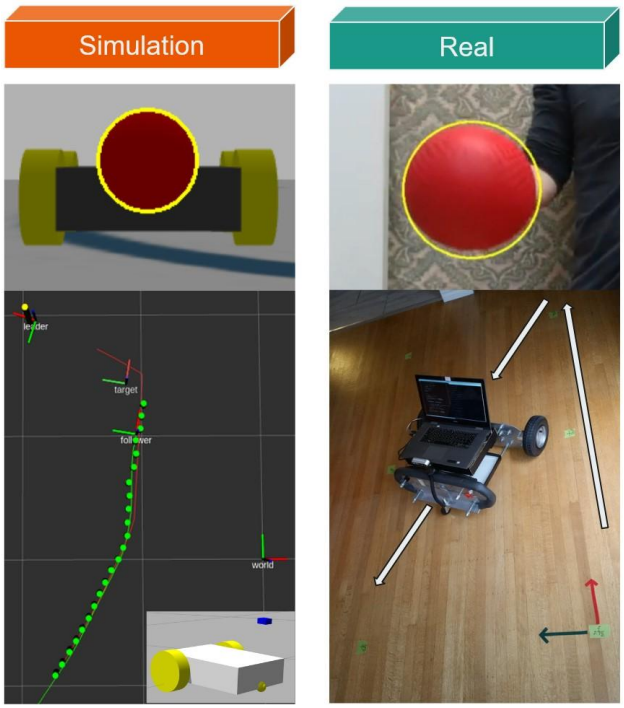
SIMULATED VERSUS REAL-WORLD DEVELOPMENT OF A SEMI-AUTONOMOUS FOLLOWING CART



Svena Yu
Simon Zheng
Shymon Sumiyoshi

Project Sponsors:
The UBC Engineering Physics Project Laboratory

Project 2054
ENPH 479 - Engineering Physics Project III
Engineering Physics Project Laboratory
The University of British Columbia
April 14, 2021

Table of Contents

Table of Contents 2

Table of Figures 3

List of Tables 4

EXECUTIVE SUMMARY 5

INTRODUCTION 6

Sponsor Objectives 6

Background 7

Visualization 8

Target Tracking 8

Controller 8

Scope and Limitations 9

Significance of Project 11

DISCUSSION 12

THEORY / FUNDAMENTALS 12

Stereovision 12

Target tracking 13

Coordinate Transforms 13

Controller movement 15

robot_pose_ekf 15

APPROACH AND DESIGN 16

Physical Robot 16

ROS Architecture for Physical Robot 19

Using the robot_pose_ekf package 21

Simulated Robot 23

Setup Details 23

ROS Architecture 24

Overview of ROS nodes and their connections in Gazebo and ROS 24

Follower State Machine 25

TESTS AND RESULTS 26

Physical Robot Tests 26

Navigation Tests 26

Edge Cases Tests 30

Simulation Tests 32

Simple Tests 32

Stress test results: 35

RECOMMENDATIONS 37

SUMMARY OF DELIVERABLES **38**

REFERENCES **38**

Appendices **39**

Mechanical Component List for Physical Robot: 39
Motor Power Wiring Diagram 40
Differential Drive Computation 40
Target Tracking Test on Carpet: 41
PID Tuning Procedure for Physical Robot 42

Table of Figures

Figure #	Figure Title	Pg
Fig. 1	Functional Pipeline	7
Fig. 2	ROS Ecosystem Diagram - Important Conceptual Elements	7
Fig. 3abc	Manual SLAM Mapping of Office Space using *rtabmap*	10
Fig. 4	Realsense Camera Overview	12
Fig. 5	Blob Detection Visualized in One Color Image Frame	13
Fig. 6ab	Diagram of Ball (Target) in Follower's Camera Coordinates	14
Fig. 7	From Relative Ball Coordinate in Camera Frame to Absolute Frame (*odom*)	14
Fig. 8	Abstract Connection of *robot_pose_ekf* in the ROS system	16
Fig. 9	Front View of Physical Robot	17
Fig. 10	Rear View of Physical Robot	18
Fig. 11	System Level Diagram for Physical Robot	19
Fig. 12	ROS Components & Linkages Used in the Physical Robot (Wheel Encoder Odometry Only)	20
Fig. 13	Physical Robot State Machine	21
Fig. 14	ROS Components & Linkages Used in the Physical Robot (EKF Package)	23
Fig. 15ab	Simulated World View in Gazebo and Rviz	24
Fig. 16	Simulation - Overview of ROS nodes and their connections in Gazebo and ROS	25

Fig. 17	Simulator Robot Following-Controller State Machine Diagram	26
Fig. 18	Real-world Square Loop Test (Carpet, Wheel Encoder Odometry Only)	28
Fig. 19	Real-world Square Loop Test (Concrete, Wheel Encoder Odometry Only)	28
Fig. 20ab	Real-world Square Loop Test Odometry and Error	29
Fig. 21ab	Target Tracking Test on Carpet	30
Fig. 22abc	Red Ball Illumination Test	32
Fig. 23abc	Red Ball Occlusion Test	32
Fig. 24abc	Yellow Ball Illumination Test	33
Fig. 25abc	Yellow Ball Occlusion Test	33
Fig. 26	Active Red Ball Detection Background Clutter Test	34
Fig. 27	Active Yellow Ball Detection Background Clutter Test	34
Fig. 28a-d	Tests with One Heading Change Ranging from 45 to 90 degrees	35
Fig. 29ab	Position of the target and projected waypoint over time for the 90 degree heading change	36
Fig. 30ab	Simulated 3 x 3 Square Loop Test, 5 Loops total	37
Fig. 31	Plot of x error and y error against time for Square Loop Stress Test	38
Fig. 32	U-turn target trajectory	39
Fig. 33	Waypoint error from target position over time for the u-turn test	39

List of Tables

Table #	**Table Title**	**Pg**
Table 1	Original Scope Table	9
Table 2	Modifications to Old Scope Table	10
Table 3	Important D435i Camera Specifications	12
Table 4	Physical Robot State Machine Description	21
Table 5	Important Dimensions Used in Simulation and the Real-World	24
Table 6	Important Depth Camera Parameters Used in Simulation and Real-World	25

EXECUTIVE SUMMARY

To reduce the amount of manual work needed to prepare for events, the Sponsor, Engineering Physics Project Lab, wants to investigate building a semi-autonomous load carrying robot led by a human target. The robot is intended to follow the path traversed by the human target. To address the most prominent problem, navigation, a physical robot and a simulation robot were built and tested. The target position was detected by both the physical and simulated robot by finding the ball attached to the target by a stereovision camera which can visualize color and depth. The physical robot can successfully follow each target position and maintain a following distance in an ideal environment. The simulated robot can successfully follow the target's continuous path, maintain a following distance and regain the path accurately despite the target occasionally leaving the field of view. Key insights are that accurate and rapid target detection is necessary such that the robot's path is close to the target's path, localization accuracy is also necessary yet real world localization methods have a tendency to drift over time. The current work adequately informs the Sponsor of a viable navigation method in an idealized environment and confirms the idea of a human target guide. To eventually transition to real world operating conditions, the current target tracking algorithm needs to be modified to be robust against occlusions, background clutter and illumination variations. The localization accuracy across larger distances needs to be improved via visual localization methods or by using extended kalman filters to perform sensor fusion.

INTRODUCTION

The Sponsor for our project is the Engineering Physics Project Lab. The staff of the Project Lab make up the core of the Engineering Physics department staff and they participate in multiple outreach events each year. This results in tiring work transporting equipment to venues around the University of British Columbia campus. To alleviate the need for this work, the Project Lab staff is interested in building an autonomous load carrying robotic cart that will be able to follow one of the staff to any location on campus.

Sponsor Objectives

The ideal final product that the Sponsor has outlined, is a semi-autonomous cart which can intelligently follow a human target while navigating in both outdoors and indoors environments around UBC while safely carrying a maximum load of 113 kg (250 lbs). The robot is intended to traverse the same path as that of the human target.

Three main objectives can be summarized from the Sponsor's ideal goals:

1. Physical Implementation of Robot
2. **Target Position Identification**
3. **Navigation Strategy and Implementation**

The last two - target tracking and navigation are the more important components and the Sponsor would like us to build a proof of concept based on these two objectives.

Background

The three main functional components of our system as established in Fig.x are Visualization, Target Tracking and Controller. These correspond to the robot's ability to perceive its surroundings and subsequently follow the target detected within its surroundings.

Functional Pipeline

Fig 1. Three main and sequential functions that feed into the next. The output from the controller feeds back into the visualization as robot localization.

We will be using an open source software framework called ROS (Robot Operating System) to build our robot ecosystem. ROS contains numerous general purpose drivers, tools, libraries that simplify complex robotic functions like trajectory planning, mapping and plumbing of devices.

ROS Ecosystem Diagram - Important Conceptual Elements

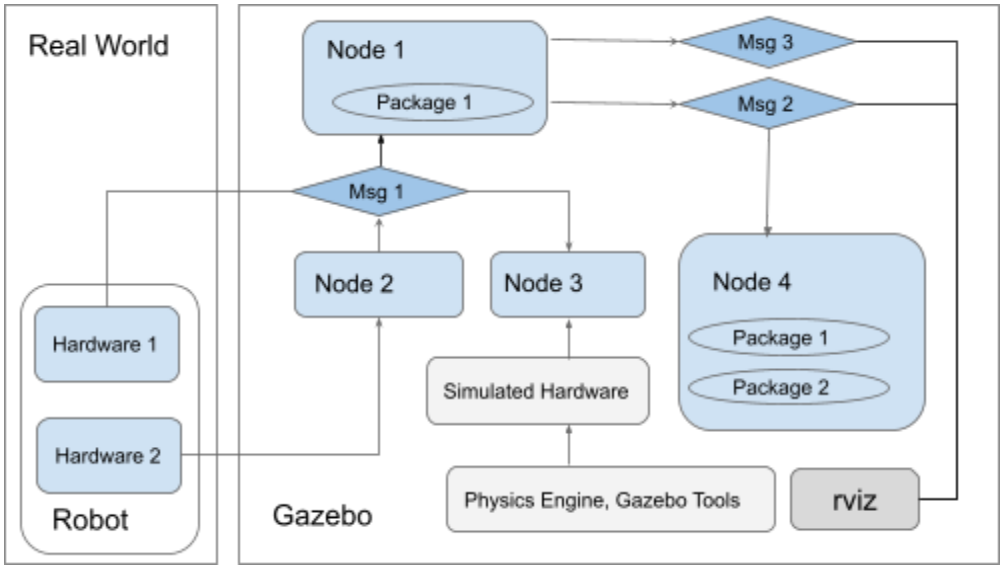


Fig 2. ROS nodes (scripts that execute various functions) can use ROS packages (collection of scripts and API to accomplish general robotic functions) and ROS communication uses ROS messages. Nodes can listen to any published messages and can interface with either the simulated hardware from Gazebo or physical hardware from the robot in the real world.

We are using two Simulation tools that are part of the ROS ecosystem. Gazebo is the physics engine that we are using to simulate the robot's behaviour and Rviz is the visualization tool that we are using to demonstrate the robot's trajectory.

Visualization

Naive detection methods are simple to implement and achieve decent performance in somewhat cluttered environments. Yang (2005)[1], and Gigliotta et al. (2003)[2] have reported viable detection methods using ultrasonic sensors or radiofrequency(RF) signals.

Less naive methods like computer vision have become dominant in the field of visualization. Stereovision can be achieved with two digital cameras and accuracy can be enhanced by a time-of-flight sensor. In the non visible light spectrum, LiDAR technology uses the scattering of waves off of surrounding objects to form a dense map of its surroundings.

As using computer vision allows greater flexibility in target tracking, path planning and localization tasks, we have chosen to implement computer vision via a stereovision camera, the Intel Realsense D435i. Binocular vision was successfully used in Neves et al. (2013)[3], and stereovision was used to follow a human target in Petrovic et al. (2013)[4], giving us a high degree of confidence in using this technology to accomplish target tracking.

Target Tracking

There are many types of target tracking algorithms and they are dependent on the type of intended target. There are two main types of tracking- one is by directly tracking the target and other is by tracking a visual aid attached to a target.

In a review by Pan et al. (2017)[5] which summarizes direct target tracking methods like particle filter and mean shift algorithm, the main objective is to accurately identify the human target within a bounding box. As we had little experience with direct target tracking, we decided to try the more straightforward approach of tracking visual aids with simple geometry.

The three common visual aids are QR code (Yi et al., 2019)[6], checkerboard[7], and other simple objects. The ball was chosen as it is symmetrical in all 3 axes, so the ball image can be independent of the target's position. It also does not need high definition at larger distances.

Controller

There are two main parts to the controller- Navigation and Actuation. Navigation encompasses path planning and obstacle avoidance. Actuation controls the movement of the robot and also

serves the double function of odometry (tracking path travelled) and localization (position relative to environment).

By restricting the robot's operational environmental conditions, we can greatly simplify our navigation approach and only include the path planning component. In the physical robot, actuation is achieved by motors and localization uses an encoder on each wheel. Additional localization tools are the IMU embedded in the camera. With a stereovision camera, we can also use visual odometry (tracking movement between frames) to help localize our robot.

Scope and Limitations

The overarching external factor that affects each stage of the functional pipeline (in Fig 1) is the current environmental conditions. In order to simplify the problem, we have restricted the operating environmental conditions of our robot.

	Hard Requirements	Soft Requirements
Target Following	● Can follow target around UBC campus ● Maintain following distance of 0.5m - 3m from target	● Good lighting only ● 250 lb load capacity robot ● Able to track target with no visual aid
Navigation Strategy	● Stops when any obstacles present or when direct line of sight to target is lost ● Can be manually controlled by user ● Clear visual aid allowed on target	● Able to re-establish line of sight ● Able to navigate around some obstacles and tricky routes
Environmental Conditions	● Good lighting only ● Few obstacles along path ● Smooth surfaces only	● Operates with some obstacles along path

Table 1. Original Scope Table

At the outset, we identified two main streams of work- one was the target tracking algorithm and the other was SLAM (Simultaneous Localization and Mapping). We intend to use SLAM to map and accurately localize the robot in this environment.

We used an open source ROS package (*rtabmap)* to visualize the mapping. Using the RealSense D435i, an office space of one of the members was successfully mapped.

Manual SLAM Mapping of Office Space using *rtabmap*

(a) (b) (c)

Fig 3. (a) Setup of Realsense camera mounted on top of a rolling chair which was slowly pushed around the office. (b) Actual Layout of Mapped Area. (c) Mapped Result in rtabmap. The black areas show borders detected.

However, we found that rtabmap required heavy computational power to run properly and only worked well within a narrow environment definition. As the barrier to working with SLAM was high and the heavy mapping component was unnecessary for our purposes (the robot was always navigating unknown environments), we concluded that SLAM was not a necessary component to add right away.

The scope table was modified to reflect more realistic and precise goals.

	Hard Requirements	Soft Requirements
Target Following	● With visual aid ● Able to maintain following distance	● Can re-establish line of sight when the leader is out of sight.
Navigation Strategy	● Smart Beeline with PID control ● No SLAM used	● Following an array of waypoints generated by locating the target ● Incorporate robot_pose_ekf package
Environment	No change.	

Table 2. Modifications to Old Scope Table

Other than not incorporating SLAM, the largest change to our scope would be the form of implementation. We decided to shift our focus from a physical load carrying cart to producing a proof of concept of navigation strategies via simulation in ROS, and a demonstration of the physical implementation via a physical robot with the relevant hardware.

Significance of Project

The critical factor in our project is that we need to concurrently track a target and navigate in an unknown environment in real time. The prevalent presence of the human target who will “guide” the robotic cart allows us to simplify our approach and is unique to most computer vision projects which involve navigation within a known environment.

Our project also serves as an important proof of concept for our Sponsor, who will use our results to inform future decisions on target tracking and navigation strategies. The physical robot demonstrates the integrated hardware suite and its performance in the presence of real world noise. The simulated robot provides a “perfect” environment to test and evaluate various navigation strategies.

DISCUSSION

THEORY / FUNDAMENTALS

Stereovision

This is specific to the Intel Realsense D435i camera we are using.

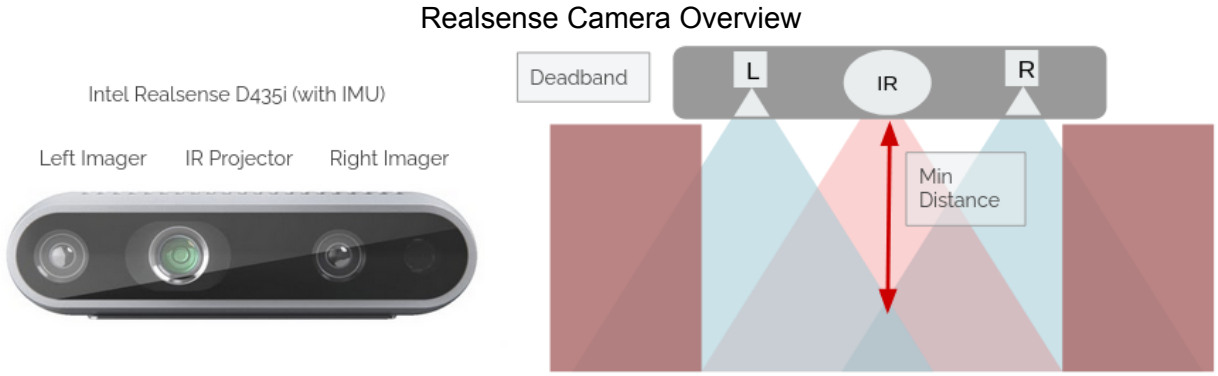


Fig 4. Realsense camera has a left/right imagers, with an IR projector in the middle which projects non-visible static IR patterns to improve depth accuracy for low texture scenes [8].Minimum depth distance and deadband areas in the depth image is as depicted.

Depth information is extracted by comparing the shift between overlapping pixels from the left and right imagers and comparing that to the scattered IR pattern received in the depth processing unit to generate a depth pixel frame. Only the overlapped image regions contain depth data, hence we expect the depth frame to have a reduced field of view.

From the D435i data sheet, these are important characteristics of our camera:

Color Stream FPS	30
Depth Stream FPS	30
Max Color Width x Height	1920 x 1080
Max Depth Width x Height	1280 x 720
Min Depth at Max Resolution	280 mm
Z Absolute Error at 2m and 80% Field of View	<= 2%

Table 3. Important D435i Camera Specifications

The minimum depth will determine the robot following distance. The Z absolute error is low even at larger distances so we have confidence in our depth readings.

Target tracking

Target tracking is accomplished by the blob detection method. The algorithm is as follows:

Blob Detection Visualized in One Color Image Frame

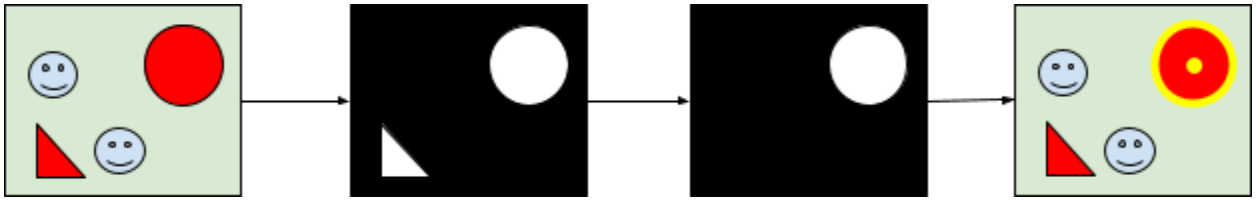


Fig 5. (From left to right) Original Image, Bit Masking, Largest Continuous Area Detected, Centroid of area and circular contour is drawn

Target tracking is accomplished in the sequence above. First, the original color image is bit masked according to custom threshold HSV/RGB values. The largest continuous area is chosen as the target area and an enclosing circle is drawn around the contours. The centroid is then found and the corresponding depth value is found within the depth pixel frame.

Coordinate Transforms

After the centroid positions are found the following coordinate transforms take place to localize the target relative to the robot's current position (assume no elevation change):

Assuming minimal distortions across the image frame:

$$Dist\_from\_Image\_Center = (Camera\_width/2 - C_x) / (Blob\_Radius / Ball\_Radius)$$

$$Rel\_X\_Coordinate = Dist\_from\_Image\_Center$$

$$Rel\_Y\_Coordinate = Depth\_Distance$$

$$Where\ C_x, C_Y = Ball\ Centroid\ Position$$

Diagram of Ball (Target) in Follower's Camera Coordinates

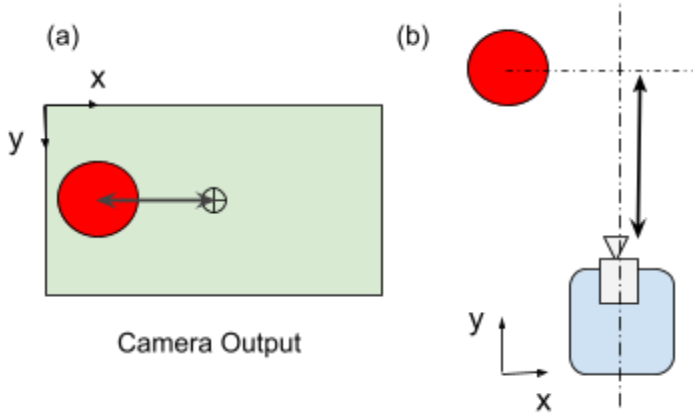


Fig 6. (a) Arrow represents *Dist_from_Image_Center*, (b) Arrow represents *Depth_Distance (from camera's depth stream)* measured from the camera's image plane

From Relative Ball Coordinate in Camera Frame to Absolute Frame (*odom*)

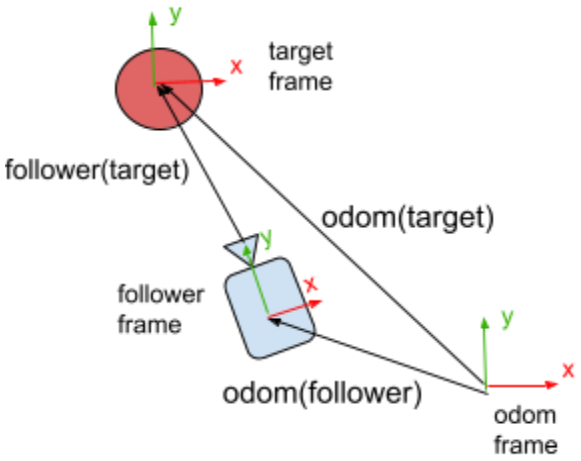


Fig. 7. Showing the *odom* frame (absolute coordinates), *follower* frame(robot coordinates), *target* frame (ball coordinates). *follower(target)* refers to the target's position in the follower frame. *odom(follower)* refers to the follower's absolute position in the world. *odom(target)* refers to the target's absolute position in the world.

From target tracking, we are able to obtain the *follower(target).* We want to obtain *odom(target)*. In ROS, frames and positions are recorded in the *pose* format.

Definition of a *pose* (position and quaternion concatenated):

Point position = {x, y, z}

Quaternion orientation = {q,x,y,z}

Point is a simple positional vector and a Quaternion is defined as:

$$Quat = cos(\theta/2) + (u_x i + u_y j + u_z k) sin(\theta/2)$$

$$Quat(q) = cos(\theta/2)$$

$$Quat(x) = u_x i * sin(\theta/2)$$

$$Quat(y) = u_y j * sin(\theta/2)$$

$$Quat(z) = u_z k * sin(\theta/2)$$

$$where\ \theta = rotation\ about\ the\ rotation\ axis\ vector\ (u_x, u_y, u_z)$$

Our rotation is about the z axis $(0, 0, 1)$, so θ can be thought of as the “yaw” angle. Vector analysis gives $odom(target) = odom(follower(target)) + odom(follower)$.

The vector, *odom(follower),* can be obtained in the real world by integrating the travelled path from the start point. It can be obtained instantaneously in simulation by simply subscribing to the */gazebo/model_states* topic.

$$odom(follower(target)) = Rot_\theta(follower(target))$$

$$Rot_\theta = q * follower(target) * q^{-1}$$

$$where\ q = unit\ quaternion = cos(\theta/2) + 1k * sin(\theta/2)$$

$$and\ q^{-1} = conjugate\ quaternion = cos(\theta/2) - 1k * sin(\theta/2)$$

We can easily adapt the above calculation to transform between multiple frames. The coordinate transforms that needs to happen:

$$camera(target) \rightarrow follower(camera(target)) \rightarrow odom(follower(camera(target)))$$

The math happens within the *tf2* ROS package, which can broadcast and listen to the *pose* of any coordinate frame in any other coordinate frame. By broadcasting *camera(target),* we are able to easily obtain *odom(target)* from the *tf Buffer.*

Controller movement

The controller needs to filter waypoints (*odom(target) pose*) set by the target tracking node and actuate the motors to bring the robot to the intended path. The path planning node simply

gathers $camera(ball)$ poses and filters them according to a threshold to form a sequence of positional goals for the robot.

Encoders count the distance travelled by the wheel by tracking the number of pulses. Each pulse indicates a small angular turn of the wheel. Hence, if the wheel experiences slippage or backlash, the number of pulses recorded will not correspond to the actual distance moved by the robot. If slippage is a systematic problem, we can see that the encoder error will increase linearly with time.

ROS Package: robot_pose_ekf

To reduce and compensate for this accumulated error in the robot's positioning, we intended to use an extended kalman filter implemented in the *robot_pose_ekf* package to obtain odometry (*/odom)* messages with higher accuracy.

Extended kalman filters are kalman filters adapted for nonlinear systems. Kalman filters use the predicted measurement and the measurement noise to extract an optimal estimated measurement. Extended kalman filters linearize a nonlinear system about the current state estimate using a first order Taylor approximation.

The *robot_pose_ekf* can be used with encoders, IMU and visual odometry. Although encoders are largely a linear system, IMU readings and visual odometry can have non linear state transitions (for example, bumps on the road, low texture surfaces).

Abstract Connection of *robot_pose_ekf* in the ROS system

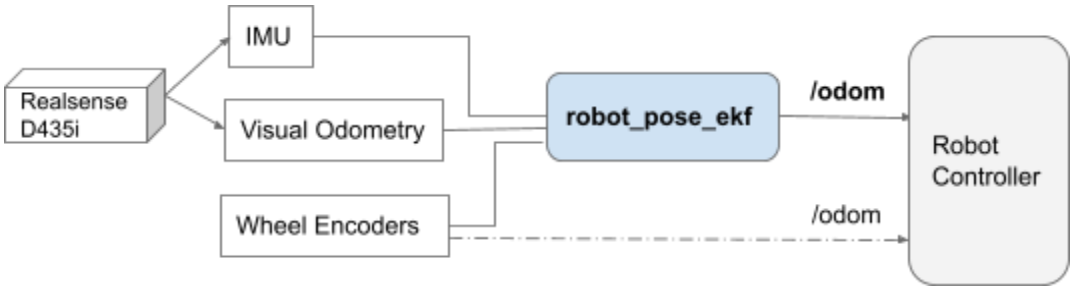


Fig 8. The *robot_pose_ekf* package can be used in the robot to replace the */odom* message from only the wheel encoders. *robot_pose_ekf* only needs two out of the three inputs shown

APPROACH AND DESIGN

Physical Robot

A two-wheeled robot with a caster wheel was used as the base platform as seen in Fig 9 and Fig 10. Refer to the *Mechanical Component List of Physical Robot* of the Appendix to view the itemized component list.

Front View of Physical Robot

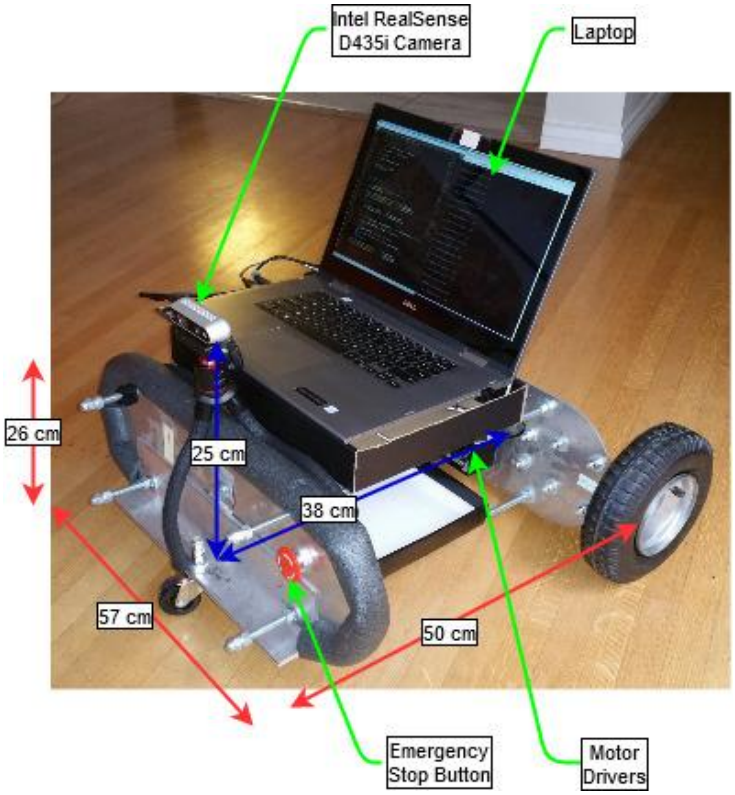


Figure 9: Important dimensions marked out in red arrows and components marked out in green.
Rear View of Physical Robot

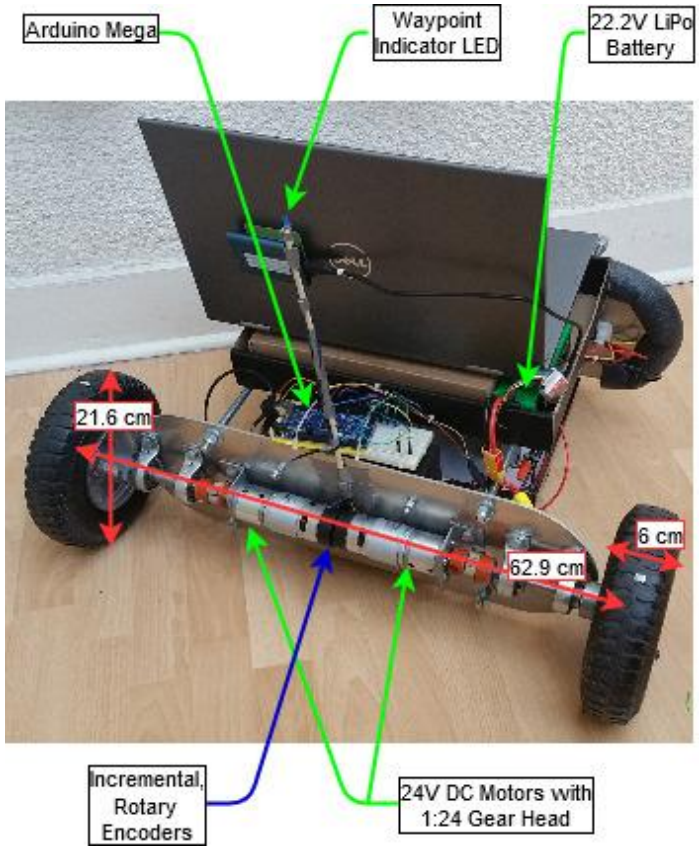


Figure 10: Important dimensions marked out in red, components marked out in green and encoders in blue.

The robot was measured and physical dimensions shown are used as critical inputs in building transform coordinate frames such that the input from the camera can be transformed accurately to actual target position relative to the origin (start point) of the follower robot.

Peripheral devices like the waypoint LED (which turns on when obtaining a new waypoint) and the Emergency Stop serve debugging and safety features which we have found are essential during the testing and operation of the robot.

System Level Diagram of Connections between Components in Physical Robot

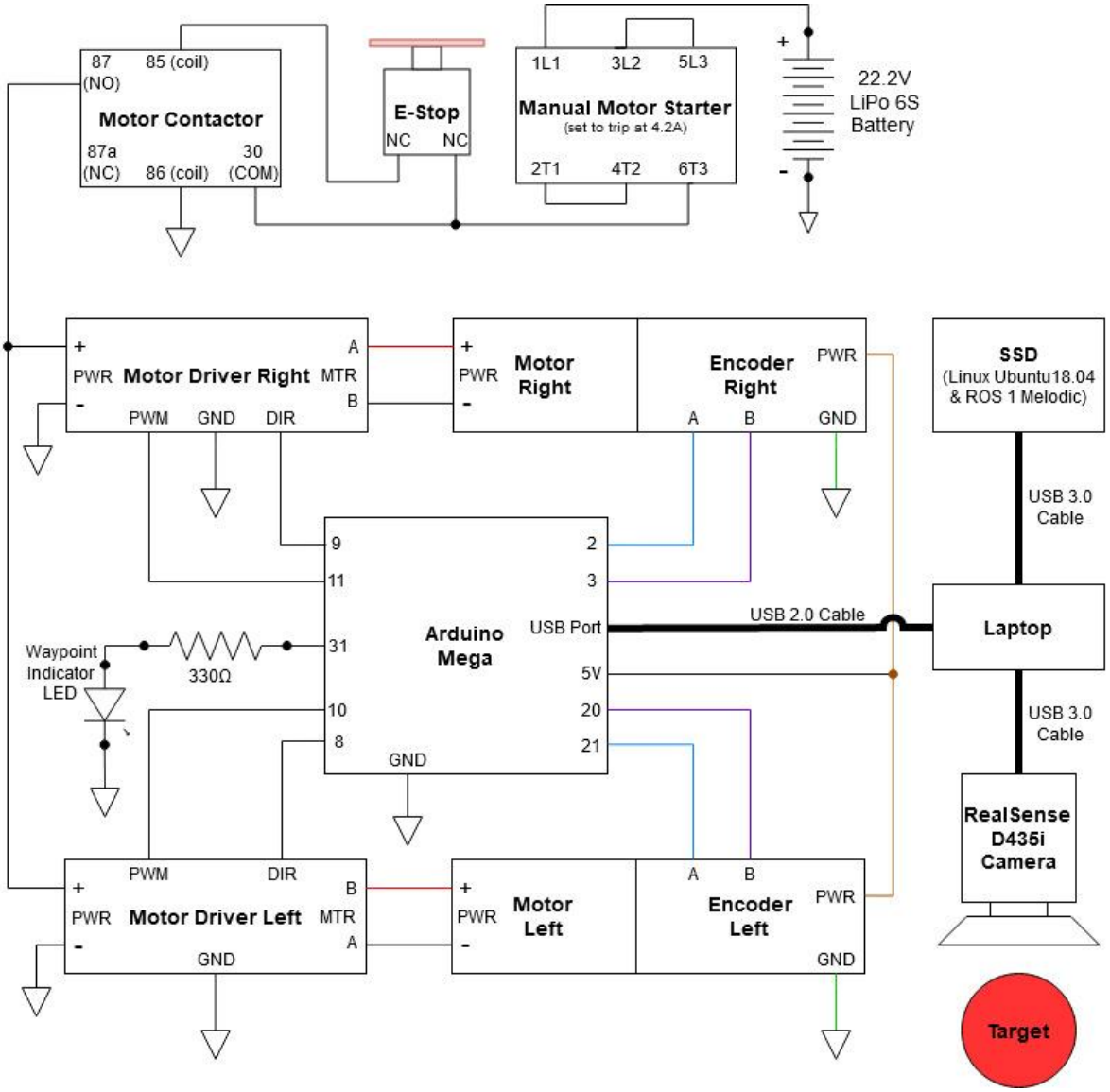


Figure 11: Standard system diagram that shows the laptop as the main computational unit of the robot. The Arduino Mega is the main Controller that interfaces with the actuation hardware on board. The Laptop is connected to both the Arduino Mega and the Realsense D435i camera.

ROS Components & Linkages Used in the Physical Robot (Wheel Encoder Odometry Only)

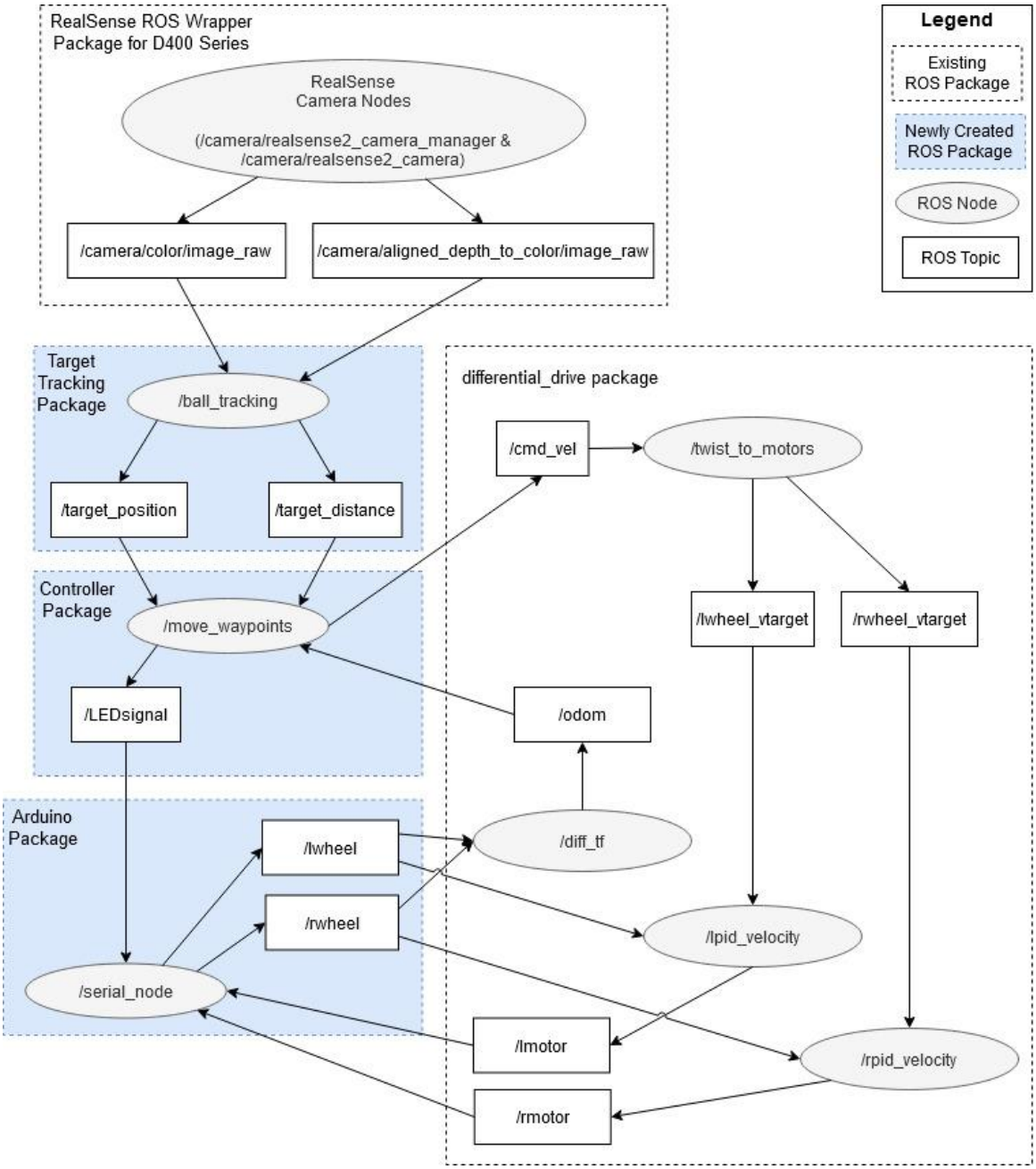


Figure 12: The *ball_tracking* node parses the depth and image stream from the target tracking node and obtains the target's position information that is fed into the *move_waypoints* node. To move to new waypoint, differential_drive package is used to send appropriate velocity commands to */cmd_vel. /lmotor* and /*rmotor* are respective commands to motor drivers. */lwheel* and */rwheel* are respective encoder counts that are fed into */odom* message and used as a positional feedback in *move_waypoints* node.

PID values found were Kp=100, Ki=75, Kd=0.0005. The procedure for finding these is listed in the Appendix under *PID Tuning Procedure for Physical Robot* (also see: http://wiki.ros.org/differential_drive/tutorials/setup).

State Machine for Physical Robot Performing Basic Target Following

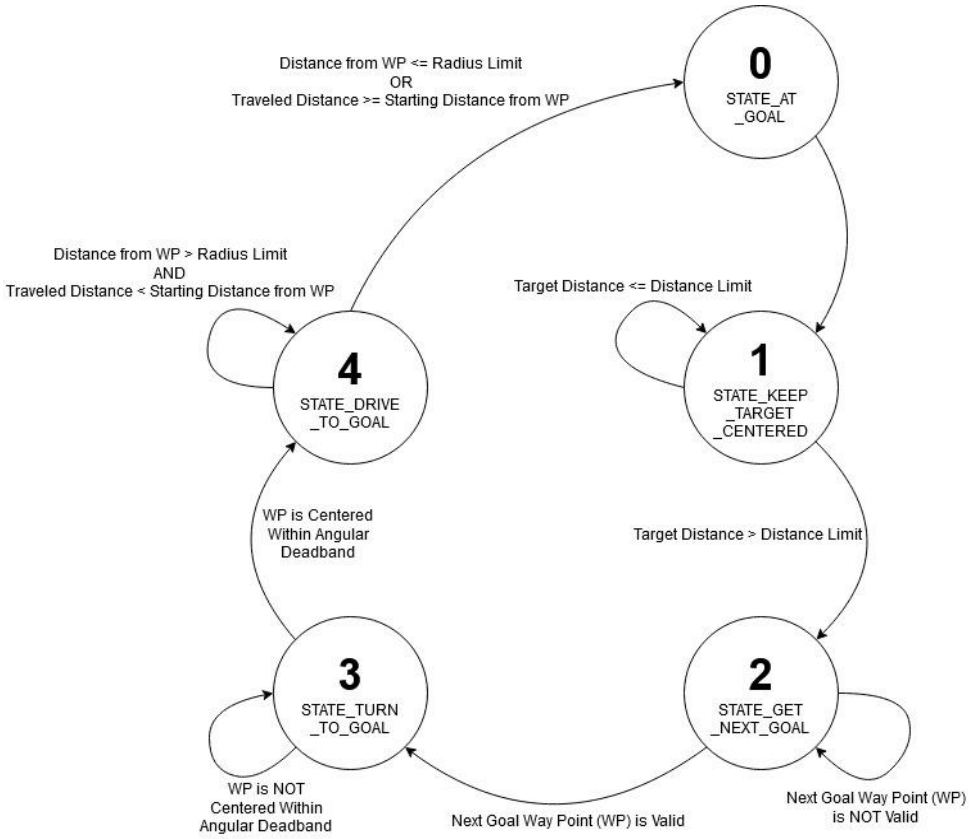


Figure 13: Physical Robot State Machine

State 0	Robot arrives at goal and stops
State 1	Robot tries to keep target centered when target distance within threshold
State 2	Target exceeds threshold distance, obtain the target position
State 3	Rotate to face the valid goal position
State 4	Go straight to reach the goal position

Table 4. Physical Robot State Machine Description

Using the robot_pose_ekf package

We tried the following inputs into the *robot_pose_ekf* package:

- Wheel encoder odometry
- Fused IMU data from the IMU Madgwick ROS package
- Visual Odometry from the rtabmap ROS package (for SLAM)

The move_waypoints node would then use this combined odometry instead of only the wheel encoder odometry as shown in figure 14.

Although, we managed to ‘connect’ the nodes in ROS, we encountered the following technical problems:

- The IMU only worked on the slower laptop but the laptop’s CPU could not keep up with the computational requirements and incorrect odometry data was obtained.
- Visual odometry from the rtabmap package used a different coordinate axis and we did not have enough time to fix this issue.

As such, we do not have actual tests to demonstrate the improvement in localization accuracy that we expect from using the *robot_pose_ekf* package.

ROS Components & Linkages Used in the Physical Robot (EKF Package)

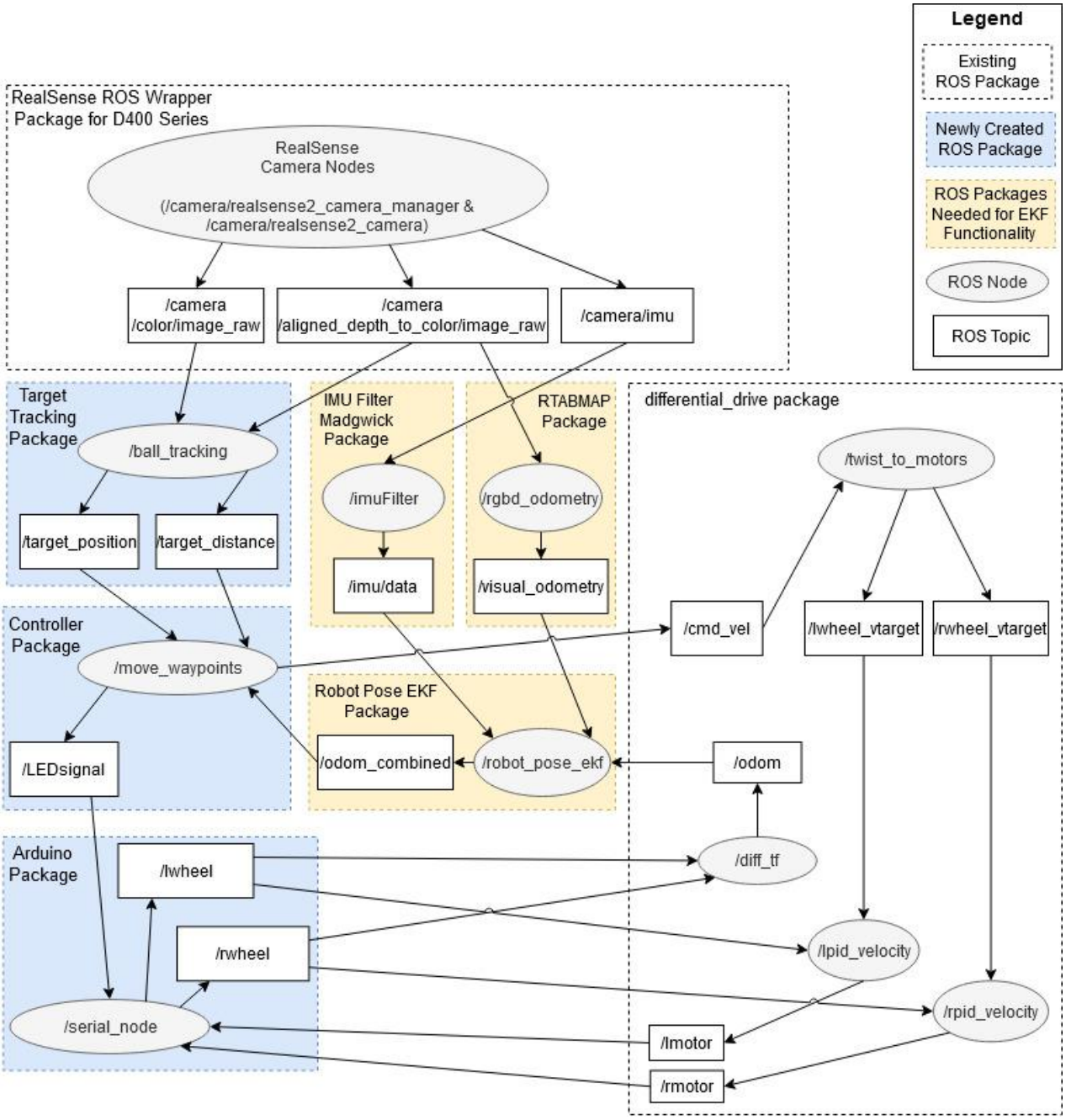


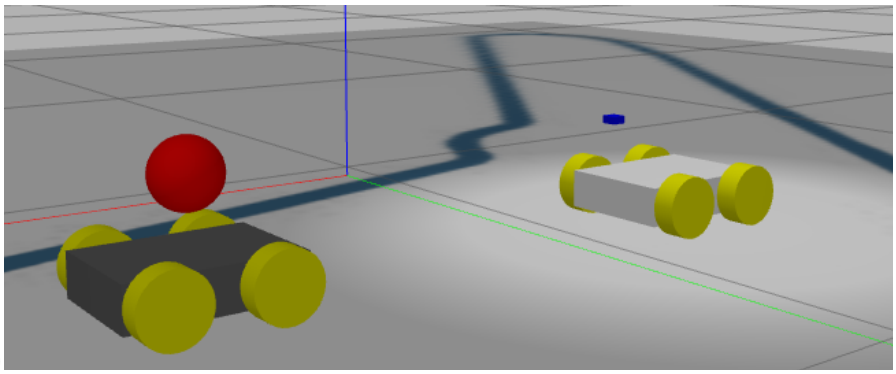
Figure 14: Physical Robot System Using the Combined Odometry from the *robot_pose_ekf* Package (Attempted Only). Difference from Wheel Odometry only encoded in light yellow.

Simulated Robot

It is no trivial matter trying to emulate the noise found in real world systems, so we have simplified our robot geometry to be a logical, movable platform with the target unobstructed by the leader geometry and the camera at the same height.

Simulated World View in Gazebo and Rviz

(a)



(b)

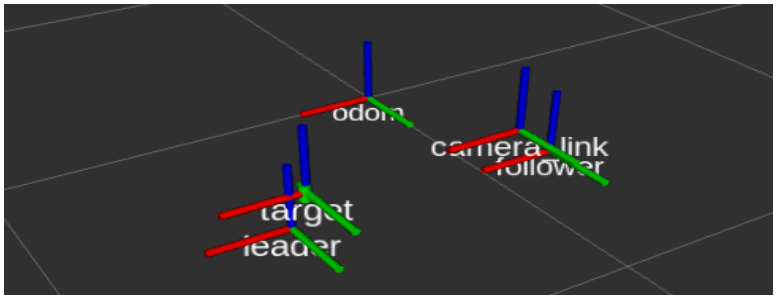


Fig 15. (a) Gazebo's simulated World View at starting positions. RGB = XYZ axis. Left is the *leader* robot carrying the ball *target* and right is the *follower* robot with a blue rectangle representing the camera (b) Simulated World View in Rviz GUI at starting positions. The *target* frame represents the ball position, *camera_link* is the frame of the camera. *Odom* represents the absolute world frame in Rviz. Each grid represents 1 unit in Gazebo.

Setup Details

For dimensions, we tried to keep the scale of the path relative to wheel diameter similar to the tests conducted with the physical robot.

	Simulation (Gazebo unit)	Physical (meter)
Wheel Diameter	0.108	0.108
Wheel Width	0.03	0.06
Wheel Separation	0.20	0.629
Ball Diameter	0.08	0.12

Table 5: Important Dimensions Used in Simulation and the Real-World

Camera Details: We are using an Open NI Kinect camera plugin in Gazebo.

	Simulation	Physical
Image Resolution	640 x 480	1280 x 720
Update Rate	30	30
Point Cloud Cutoff	0.28	0.28

Table 6. Important Depth Camera Parameters Used in Simulation and the Real-World

With a high image resolution at 1280 x 720 in the simulation, the detection rate was around 5 Hz. Reducing the resolution to 640 x 480 maximizes effective detection rate to about 8-10 Hz without affecting navigation performance significantly.

ROS Architecture

Overview of ROS nodes and their connections in Gazebo and ROS

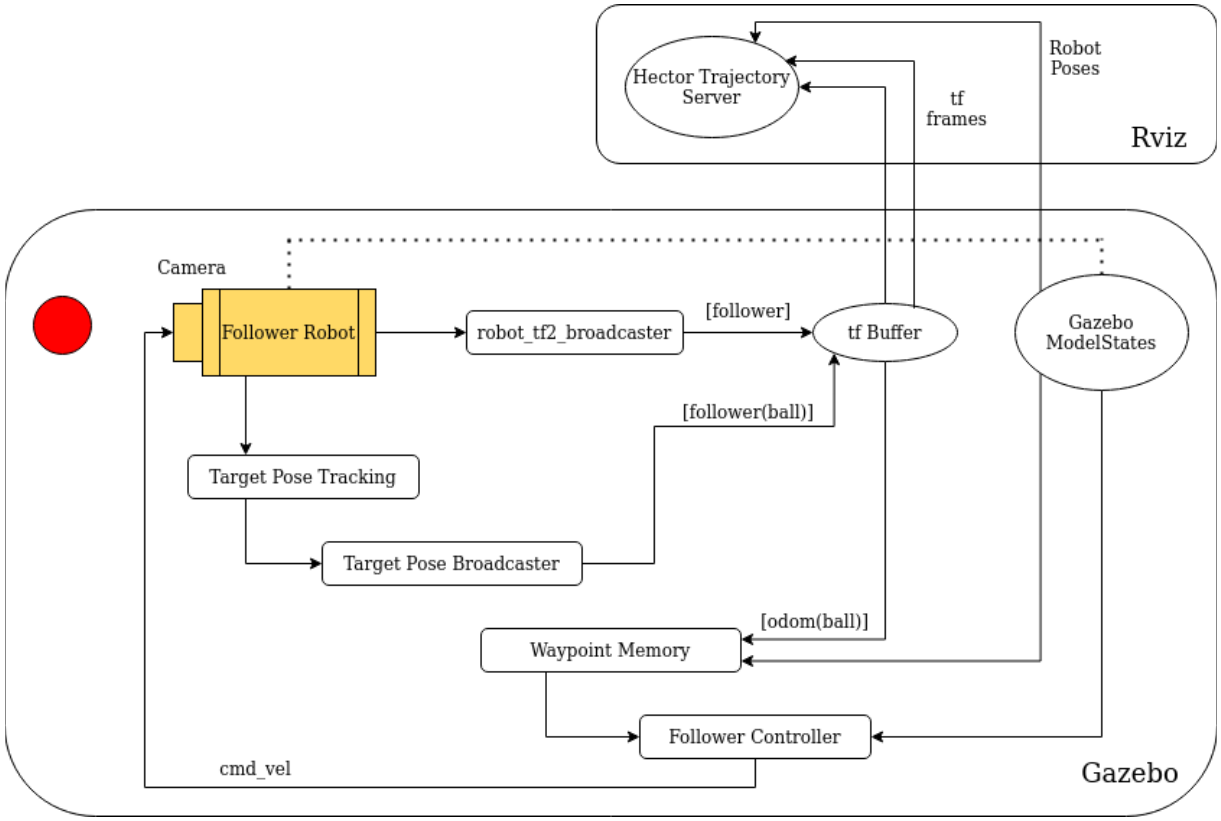


Fig 16. Rectangles represent custom ROS nodes. Ellipses represents inherent ROS functions or incorporated ROS packages. Coordinate transform (tf) frames represented by [parent(child)] are sent to the *tf Buffer* and used by other nodes. *cmd_vel* is the velocity command sent by the follower controller node. Rviz visualizes the *poses* and *tf* frames.

The four main functional nodes:

- Target Pose Tracking: **Implements** target tracking algorithm
- Target Pose Broadcaster: **Broadcasts** *camera(target) pose* to the *tf Buffer*
- Waypoint Memory: Obtains *odom(target)* and **filters** poses to add to “path”
- Follower Controller: Implements a **state machine** to follow the path and handle the scenario where the leader is out of view.

The *robot_pose_ekf* package was integrated into the simulation but there was no change as the encoder and IMU inputs used were perfect with no drift.

Follower State Machine

Follower State Machine Diagram (Handle Leader Out of View Scenario)

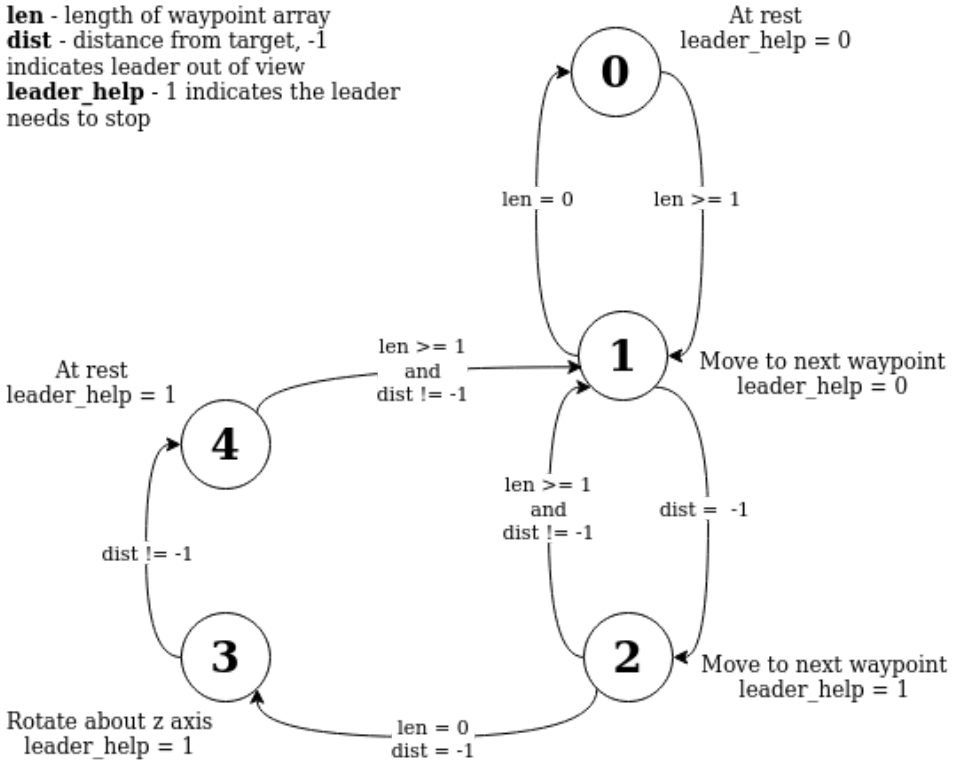


Fig 17. (State 0) Robot is stopped. (State 1) Robot is following sequential waypoints and target is in view. (State 2) Still following waypoints but target is not in view. (State 3) No more waypoints and target not in view. (State 4) Seen the target, stop to store new waypoint(s).

TESTS AND RESULTS

The main purpose of our tests is to evaluate the extent to which the scope has been fulfilled and to find current performance limitations such that future work will be informed on which aspects need to be improved.

Based on the paper *Online Object Tracking: A Benchmark*, Wu et al. (2013) [10], we have adapted performance attributes mentioned to our project scope. Relevant edge cases:

1. Illumination Variation
2. Occlusion
3. Background Clutter

For physical tests, we focused on characterizing the physical robot's actuation and for simulation tests, we tried to characterize the limits of our current navigation strategy.

Physical Robot Tests

The robot was brought into a wide, open office space with good lighting. Waypoints were measured out, marked and programmed into the robot. After each loop around the rectangle the actual position of the robot was measured. Odometry values are plotted in the graph with the actual positions overlaid. This test was repeated on two surfaces- concrete (more slippery) and carpet (less slippery) to see if real world friction had an effect on the accuracy of navigation.

Navigation Tests

5 Loops test on **carpet** with wheel encoder odometry only

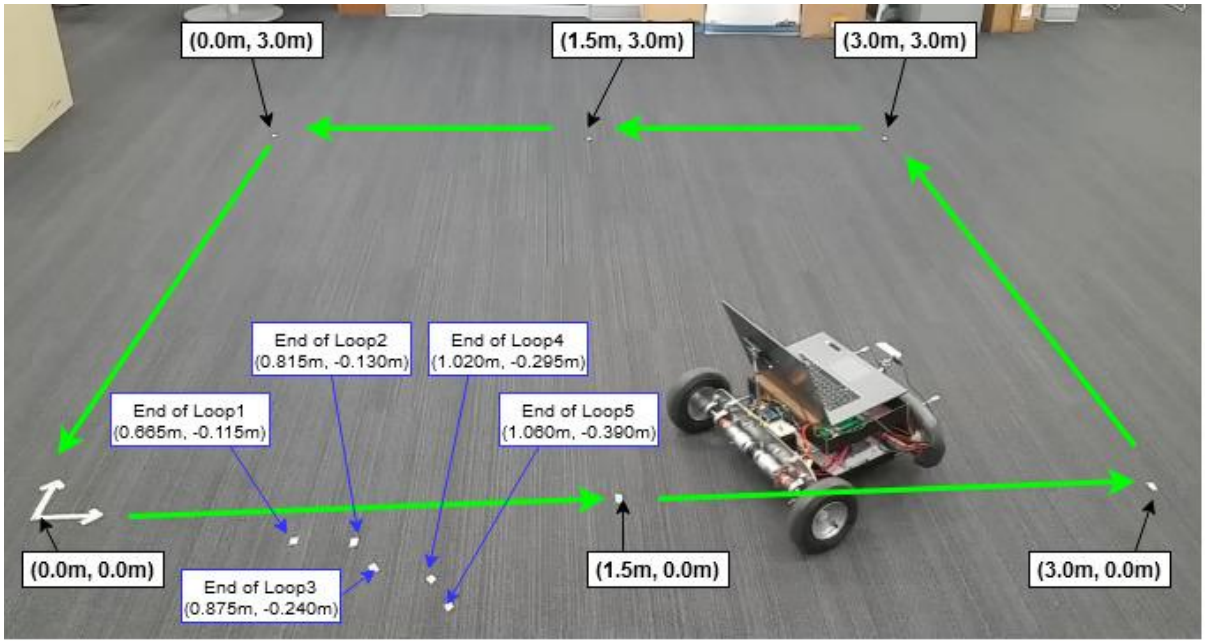


Figure 18: The waypoints are marked out on the carpet and the ending positions of each loop is marked out and measured. Full Video: https://youtu.be/pYhdFmPy1Ro

5 loop test on **concrete** with wheel encoder odometry only.

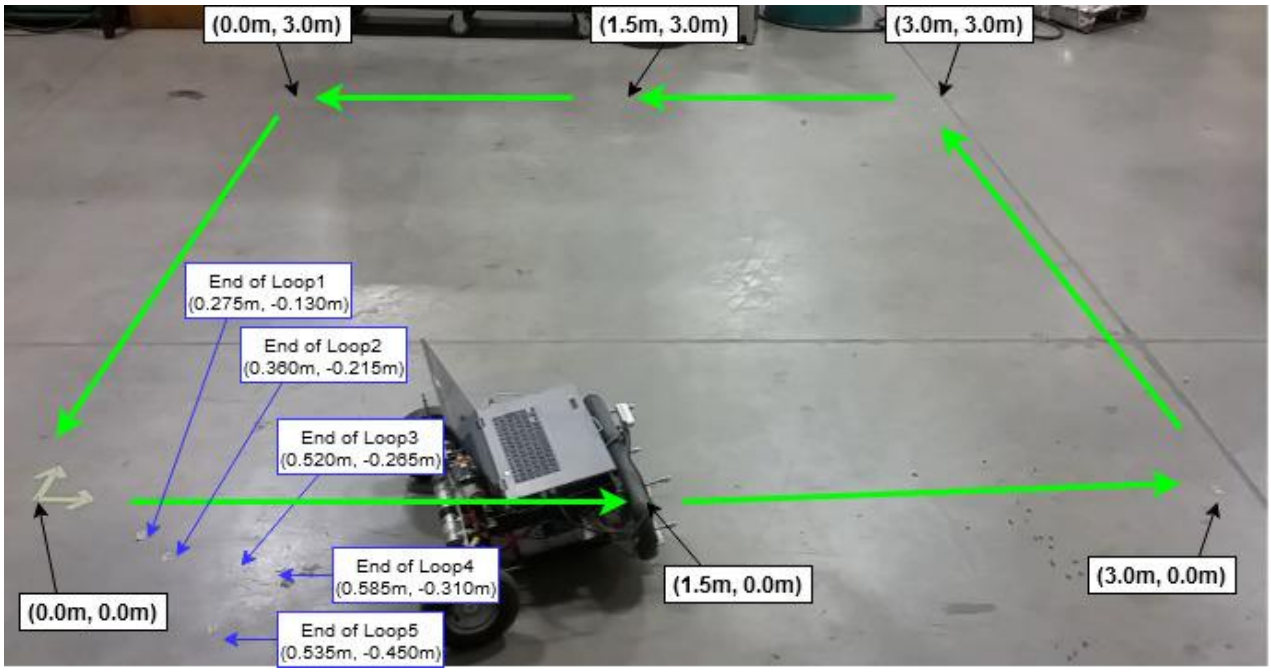
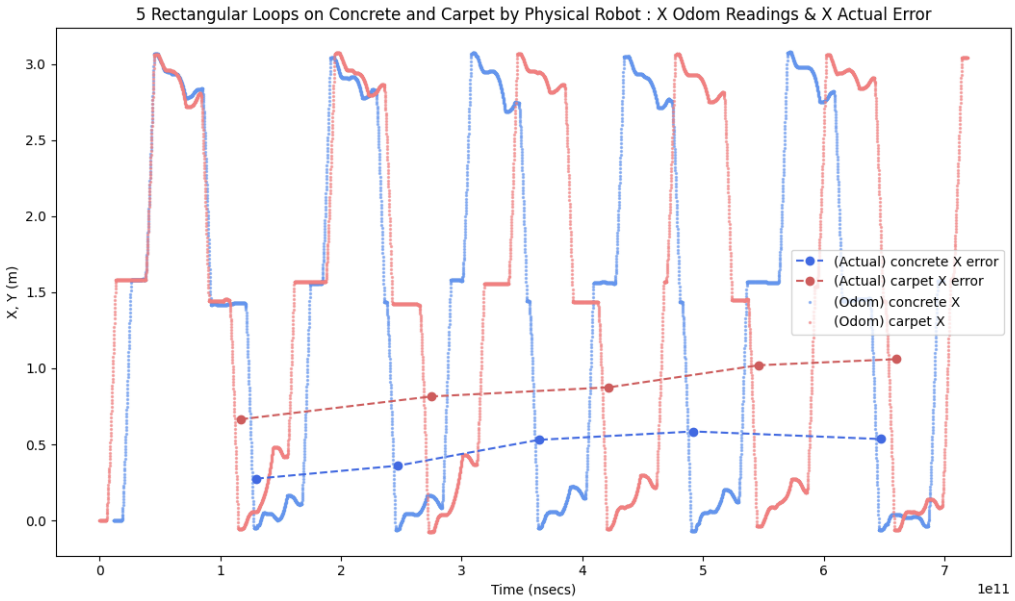


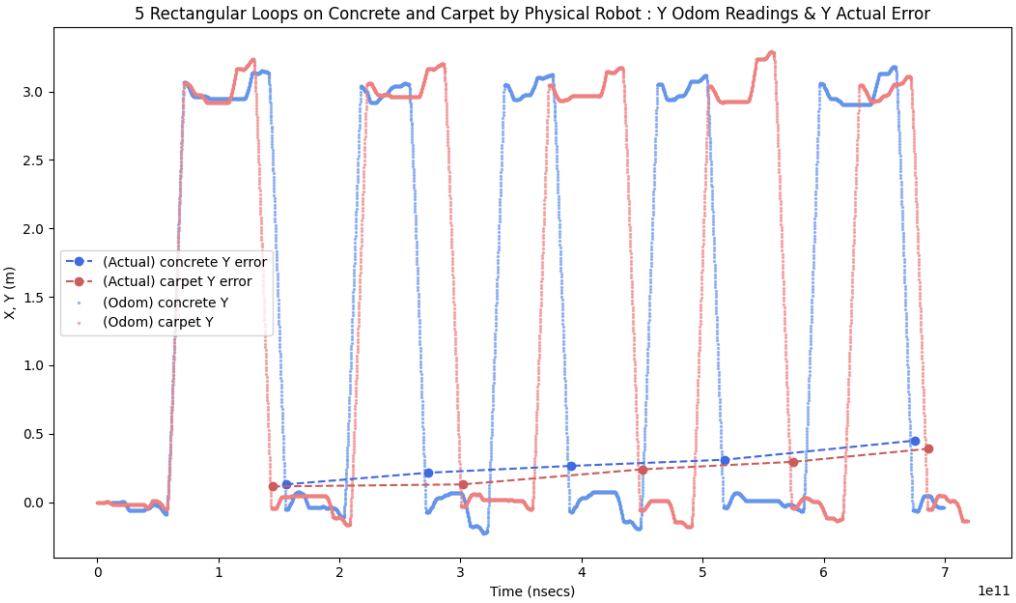
Figure 19: The waypoints are marked out on concrete and the ending positions of each loop is marked out and measured. Full Video: https://youtu.be/dik9-0XMeEU

As shown in Fig 20a) and 20b), we found that the error between the robot's actual position and its assumed position increased with each loop. From the videos, we can also observe how the robot's rectangular path appears to slowly rotate counter clockwise along each loop.

X and Y Odometry Data for Carpet and Concrete Tests & Absolute Actual Positional Error



(a)



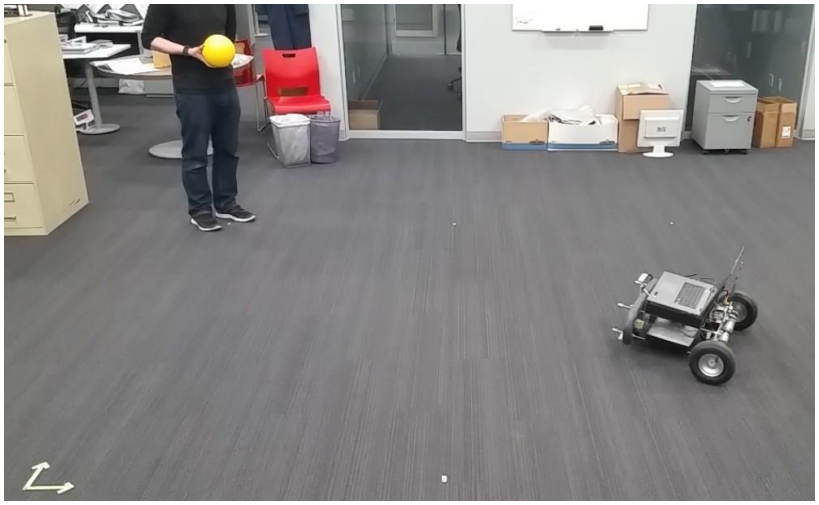
(b)

Fig 20. (a) X data from Odometry data is plotted against time. Overlaid plot of absolute X position error at the end of each loop. (b) Y data from Odometry data is plotted against time. Overlaid plot of absolute Y position error at the end of each loop. The end of a loop is indicated by the return to "0" position.

We can deduce that the encoder error increases almost linearly with distance travelled. This can be seen as the (almost) straight line error as the number of loops traversed increases. Due to the need to start and stop 6 times in the X (vs. 4 times in Y), the absolute positional error is larger in the X direction and the difference between the two surfaces is more apparent due to the larger static friction coefficient difference between carpet and concrete.

Target Tracking Test on Carpet

(a)



(b)

Fig 21. (a) The robot has acquired a new waypoint and is in State 4, driving to the new waypoint. (b) The robot is trying to acquire a new waypoint and is in State 1, rotating on the spot to find the ball. (Picture at 4min30sec in Full Video: https://youtu.be/3xi6AV5KmyQ)

All the states (State 0 - State 4) have been demonstrated in the video and for the timing breakdown, refer to the section *Target Tracking Test on Carpet* in the Appendix. The state machine appears to work well. The robot can follow each new waypoint and maintain a following distance of 2m.

We found that the robot was able to adequately follow the target in ideal conditions (bright, consistent lighting and level, smooth terrain). The following observations were made:

- Wheel velocity PID values need to be further optimised to ensure faster respons
- Slower target wheel velocities are better to minimize overshoot of linear and angular velocities and also increases accuracy of IMU/encoder values obtained.
- Target tracking algorithm needs to be refined to reduce likelihood of false positives. Yellow objects in the environment in the video were hidden prior to the test.

Edge Cases Tests

1. **Illumination & Occlusion Tests**

   We tested the effect of illumination variation on target tracking and the maximum extent to which the target could be occluded before target identification was no longer possible. HSV values used were for ‘night time’ lighting conditions. Video for red ball test: https://youtu.be/Xx6Zt7vXYqM. Viideo for yellow ball test: https://youtu.be/ShDGrK-kzvk

Red Ball Illumination Test

Figure 22: Red ball target identification in good (left), dim (centre), and dark (right) lighting with NO target occlusion. Yellow circles become smaller and encapsulate less of the ball as lighting quality decreases.HSV value range from (0,129,127) to (181,255,255)

Red Ball Occlusion Test

Figure 23: Red ball target identification in good (left), dim (centre), and dark (right) lighting WITH target occlusion. Pictures show the maximum extent to which the target can be occluded before the yellow hough circle disappears. In good, dim, and dark lighting, target can be occluded by approximately 80%, 50%, 20% respectively before the target tracking node can no longer detect the target. HSV value range from (0,129,127) to (181,255,255)

Yellow Ball Illumination Test

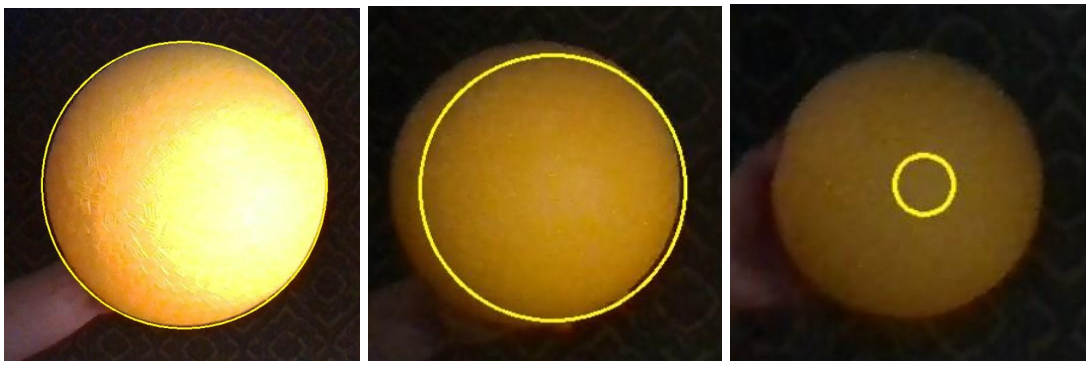


Figure 24a,b,c: Yellow ball target identification in good (left), dim (centre), and dark (right) lighting with NO target occlusion. Yellow circles become smaller and encapsulate less of the ball as lighting quality decreases. Yellow HSV Range: (12,110,123) to (26,255,255)

Yellow Ball Occlusion Test

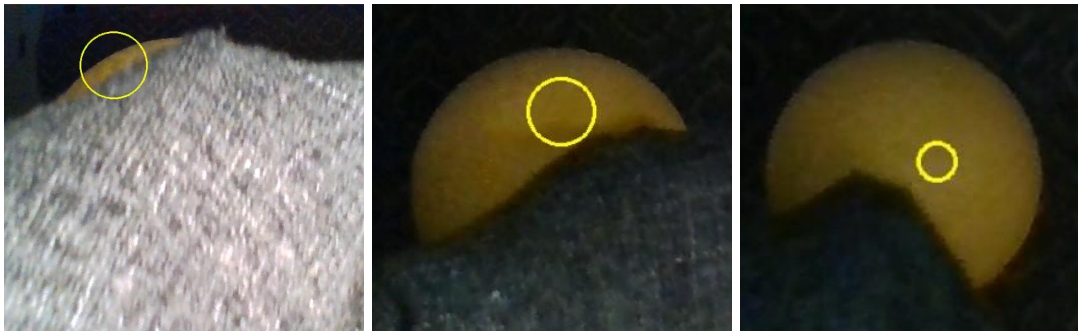


Figure 25a,b,c: Yellow ball target identification in good (left), dim (centre), and dark (right) lighting WITH target occlusion. Pictures show the maximum extent to which the target can be occluded before the yellow hough circle disappears. In good, dim, and dark lighting, target can be occluded by approximately 95%, 55%, 25% respectively before the target tracking node can no longer detect the target. Yellow HSV Range: (12,110,123) to (26,255,255)

We found the yellow ball was more easily detected in different lighting and could be occluded noticeably more than the red ball in good lighting before target detection was lost. However, there wasn't much difference in occlusion results in dim and dark lighting.

2. **Background clutter**

To demonstrate how the visual noise caused by background clutter affects target tracking, two tests were recorded with background elements with red and yellow hues.

Active Red Ball Detection Background Clutter Test

Fig 26. (left) Two balls apart, detects well. (center) Two balls together, contour inaccurate. (right) Ball on the side, contour inaccurate.

Active Yellow Ball Detection Background Clutter Test

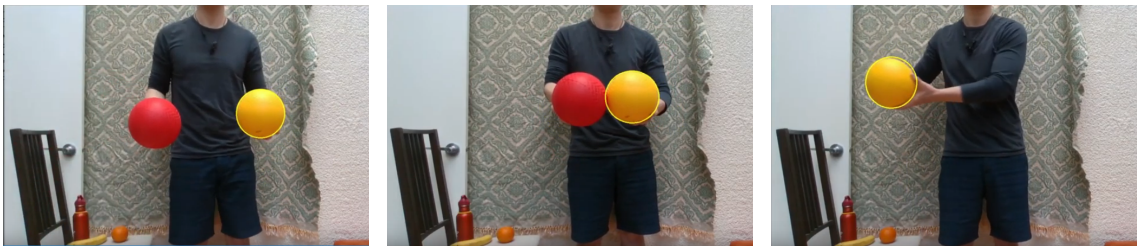


Fig 27. (left) Two balls apart, detects well. (center) Two balls together, detects well. (right) Ball on the side, detects well.

We can see that detecting the yellow ball is more robust than the red ball detection. However, this algorithm needs to be improved to be used practically in the real world, where the background clutter cannot be controlled. Attempts at using Hough Lines algorithm (detecting straight lines in image) to ignore rectangular blobs have been successful. Perhaps Hough Circles could be used to filter out non circular contours in the image frame.

Simulation Tests

Simple tests:

1. Straight. Rotate 45 deg clockwise. Head straight
2. Straight. Rotate 60 deg clockwise. Head straight
3. Straight. Rotate 75 deg clockwise. Head straight
4. Straight. Rotate 90 deg clockwise. Head straight

Stress tests:

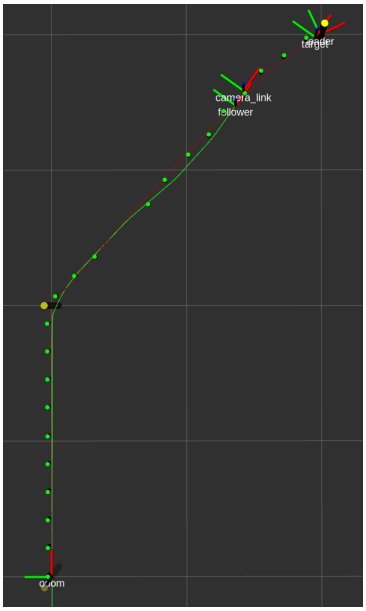
5. Rectangle
6. U-Turn

Results are shown in the visualizer, Rviz. The trajectory of both the target (red) and the robot (green) are also displayed. The robot's waypoints are green markers and the targets' yellow.

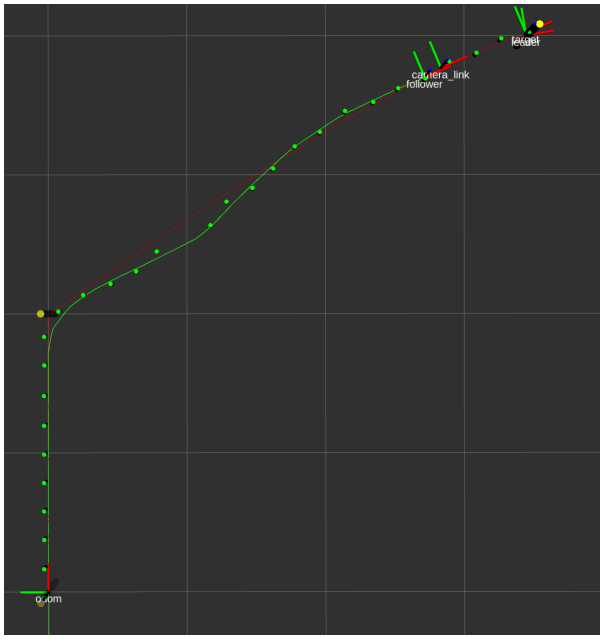
Simple Tests

The following tests demonstrate the robot's ability to track and follow the target through a single direction change with increasing angle.

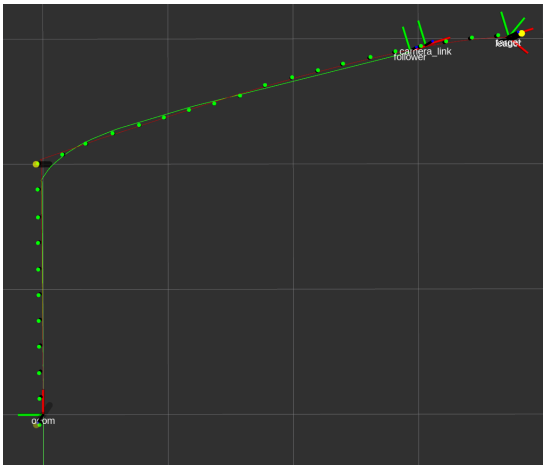
Tests with One Heading Change Ranging from 45 to 90 degrees



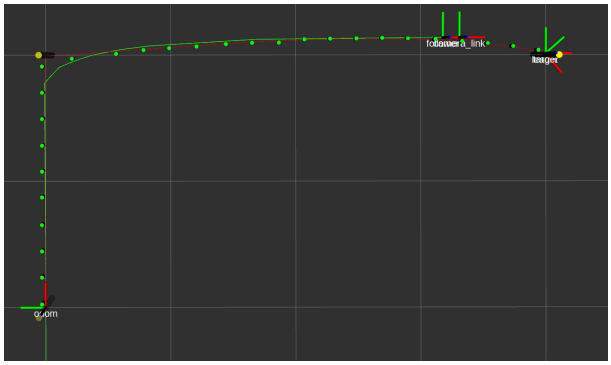
a) 45 degree target heading change



b) 60 degree target heading change



c) 75 degree target heading change

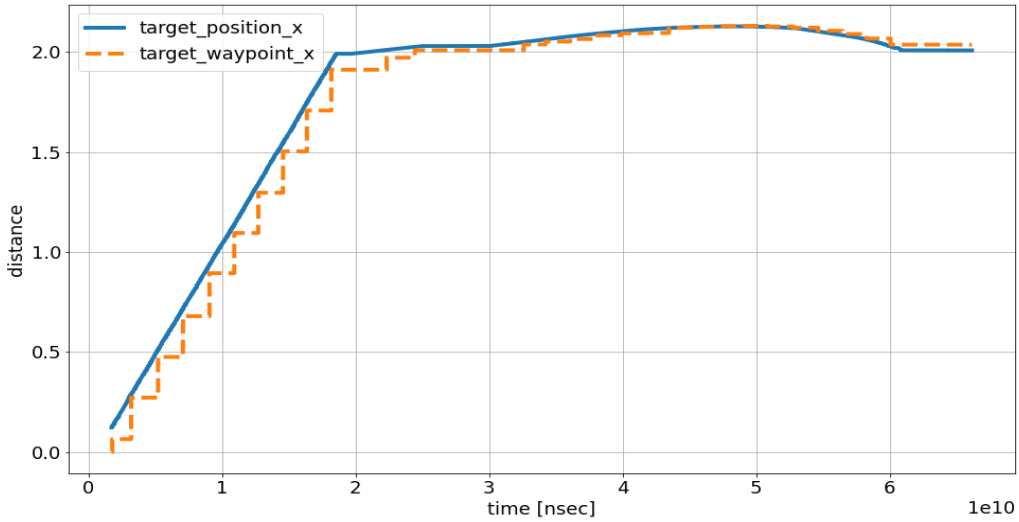


d). 90 degree target heading change

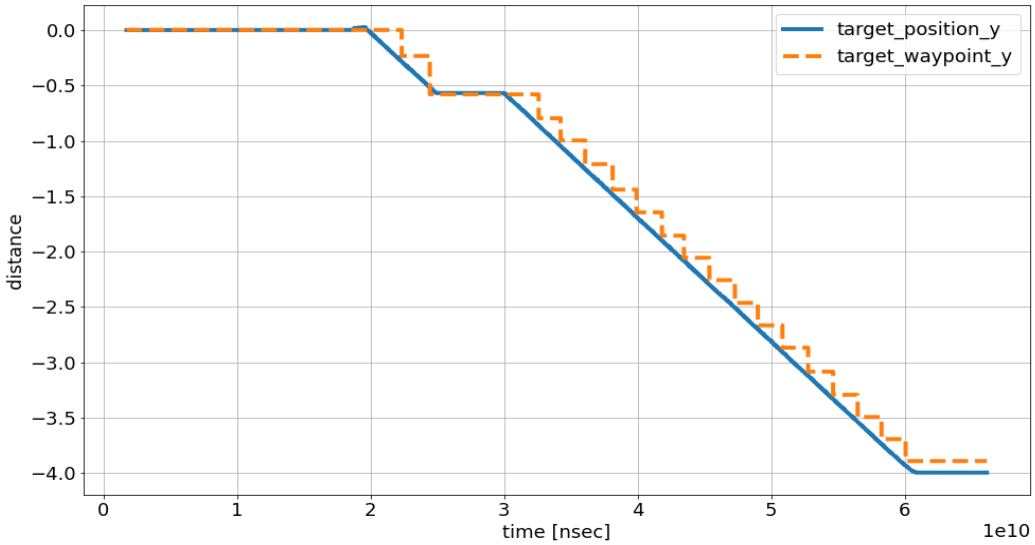
Fig 28. The waypoints closely match the actual target path in each test, as can be seen between closeness between the solid line and the “dotted” line

For large heading angle changes (eg. 75, 90deg), The target quickly exits the robot’s field of view. This discretization of the leader position leads to the steps visible in both plots.

Position of the target and projected waypoint over time for the 90 degree heading change



a) X axis



b) Y axis

Fig 29. (X,Y) coordinates of the target’s actual position against the waypoint position generated through the target-tracking pipeline. The steps correspond to the minimum distance between new waypoints (~0.3). Around the transition point (t = 2e10ns), a new waypoint is not generated for a longer period of time due to losing line-of-sight to the target.

Based on these results, it is likely that the robot can accurately track the leader's motion through a sequence of basic moves.

Stress test results:

A square-loop test was mimicked in simulation which is the worst-case scenario since the robot will lose sight of the target at each corner. Over repeated laps, outlier's begin to form, likely due to the image processing rate decreasing from 8Hz to 5Hz. The pipeline between nodes in ROS does not have deterministic behaviour, resulting in such erratic behaviour.

Simulated 3 x 3 Square Loop Test, 5 Loops total

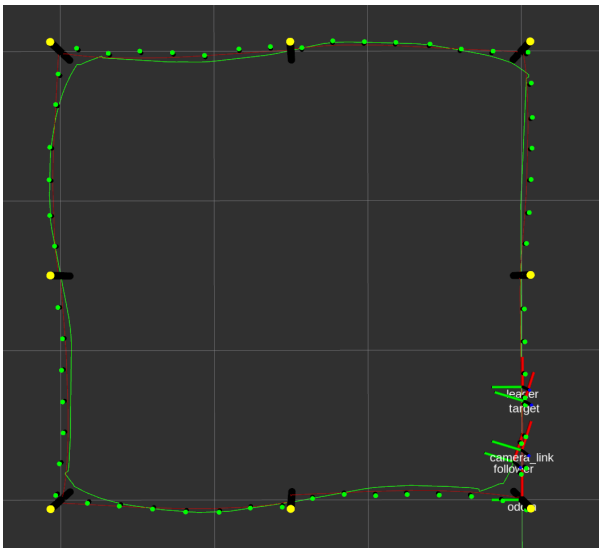


Fig 30. a) First loop

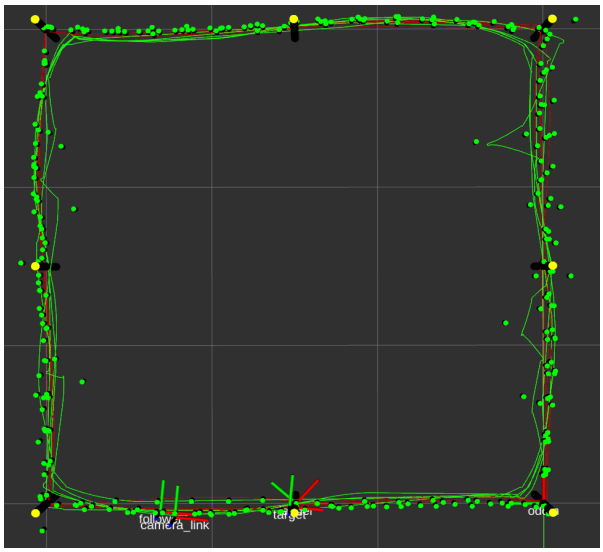


Fig 30. b) 5th loops

Plot of x error and y error against time for Square Loop Stress Test

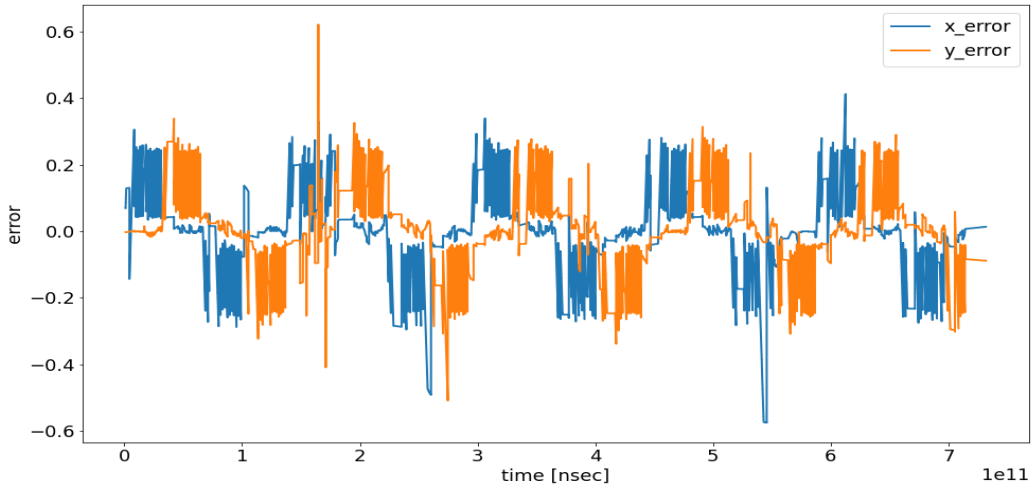


Fig 31. We can see that there are occasionally large fluctuations but both x and y distance show no individual bias and the average error remains roughly the same throughout.

The error over time (Fig. 31) shows that these extreme points occurred at various times through the test and did not significantly affect the overall target-following performance, demonstrating the controller's robust behaviour.

The U-turn test emulates the most severe heading change the target can undergo (180 degrees). While the error is similar to the square-loop test, it takes much longer for the robot to handle the maneuver as it has to visually reacquire the target often.

U-turn target trajectory

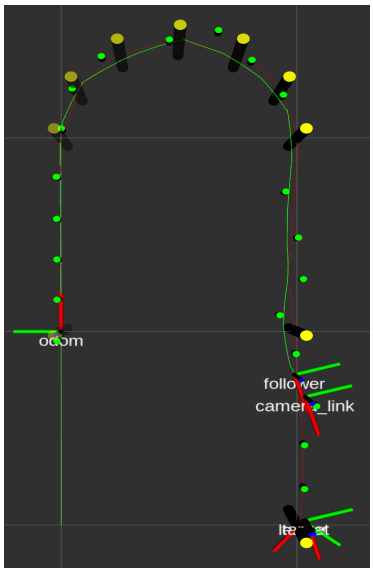


Fig 32. The waypoints deviate more near the end of the test. U turn has 0.5 unit radius curve

Waypoint error from target position over time for the u-turn test

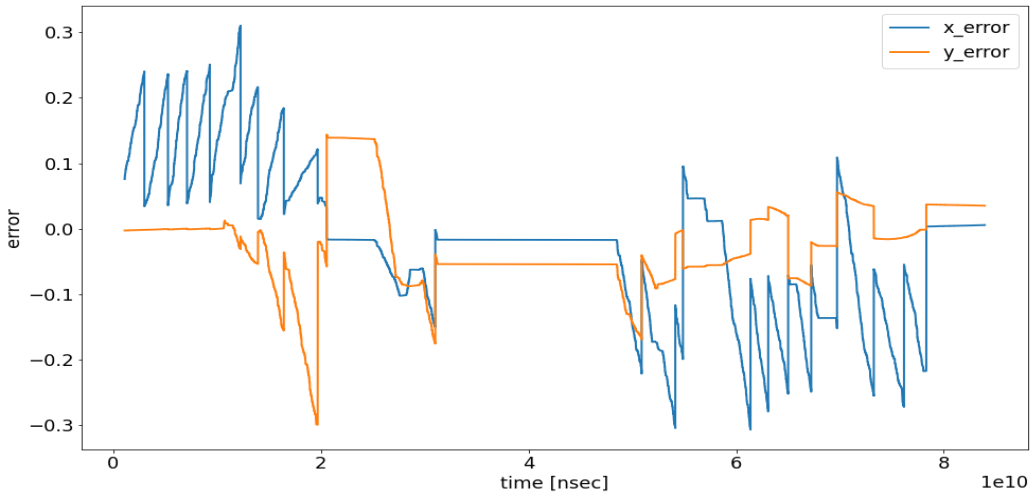


Fig 33. (x, y) error between projected waypoints and actual target location

CONCLUSIONS

The Sponsor's original objective was to investigate a semi-autonomous cart that can follow a staff around campus. In order to address the Sponsor's main concern regarding navigation, we built a physical robot and a simulation robot as a proof of concept.

Based on our test results, we have demonstrated the physical robot is sufficient to fulfill the hard requirements of the scope table and the simulated robot has fulfilled the soft requirements. As the target tracking algorithm shows good performance in simulation and in the physical robot, we expect the navigation strategy currently implemented in simulation to work similarly on the physical robot, therefore achieving the soft requirements as well.

A significant risk is that we do not have empirical tests proving that the *robot_pose_ekf* package can give us the odometry accuracy we need to navigate long distances. There current visualization method using blob detection on a ball is prone to occlusion and illumination problems. A more robust algorithm will likely need to be developed. Through our tests, we learnt that accurate target tracking and high processing speed were the main factors in improving navigation performance of the robot.

The current state of work shows that blob tracking a ball as a visual aid in idealized environments works well even and is able to move along the path set by the target even when direct line of sight to the target is occasionally lost, hence validating the Sponsor's initial navigation ideas.

RECOMMENDATIONS

The immediate next step should be focused on derisking the localization issue by incorporating the *robot_pose_ekf* into the physical robot and testing for longer on different surfaces. The target tracking algorithm should also be made more robust to withstand changes in illumination and noise from background clutter. The current state of the art for target tracking should be concurrently investigated for robust methods optimized for speed and tracking at a distance. Human pose detection like in Koide et al. (2018)[11] are commonly used and may serve as a more reliable algorithm. Open source packages like *OpenPose* can be used to investigate this method.

In addition, image processing is the computationally heavy part of the functional pipeline and speeding this process up by means of a dedicated computer like Nvidia's Jetson Nano, which has a more powerful GPU (Graphics Processing Unit) than a standard Raspberry Pi, could greatly improve performance.

As the project advances, the ability to detect obstacles and intelligently avoid static and dynamic obstacles will become necessary. Localization and local mapping (local SLAM) will be critical for complex trajectory planning. It is our recommendation to first search for all available ROS packages before testing any in detail. Most of the relevant packages (eg. *husky_navigation, costmap_2d, gmapping*) require a functional ROS Navigation Stack so it is also our recommendation to set up the Nav Stack prior to further advances in the trajectory planner.

SUMMARY OF DELIVERABLES

1. Physical Robot with Hardware Suite (excluding laptop)
2. Simulation Robot in the form of code. Available here by 30th April 2021: https://github.com/miyesven/SmartCarts
3. Logbook

REFERENCES

1. Yang, Chuan-Hao. (2005). A Person-Tracking Mobile Robot Using an Ultrasonic Positioning System. 71.
2. Gigliotta, O., Caretti, M., Shokur, S., & Nolfi, S. (2005). Toward a person-follower robot.
3. R. Neves and A. C. Matos, "Raspberry PI based stereo vision for small size ASVs," 2013 OCEANS - San Diego, San Diego, CA, USA, 2013, pp. 1-6, doi: 10.23919/OCEANS.2013.6741334.
4. Petrović E, Leu A, Ristić-Durrant D, Nikolić V. Stereo Vision-Based Human Tracking for Robotic Follower. International Journal of Advanced Robotic Systems. May 2013. doi:10.5772/56124
5. Pan, Z., Liu, S. & Fu, W. A review of visual moving target tracking. *Multimed Tools Appl* 76, 16989–17018 (2017). https://doi.org/10.1007/s11042-016-3647-0
6. (QR) Simulation and Experiment on Artificial Landmark-Based Monocular Visual Navigation System for Mobile Robot", Hu, Xu, Yang, Yi, 2019
7. (Checkerboard) Andrey-Leshenko. Andrey-leshenko/chess_6dof. Retrieved April 10, 2021, from https://github.com/andrey-leshenko/chess_6dof/blob/master/docs/guide.md
8. Intel Realsense D400 Product Family Datasheet. (2019). Retrieved April 10, 2021, from https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf
9. Hacene, N., Mendil, B. Behavior-based Autonomous Navigation and Formation Control of Mobile Robots in Unknown Cluttered Dynamic Environments with Dynamic Target Tracking. *Int. J. Autom. Comput.* (2021). https://doi.org/10.1007/s11633-020-1264-x
10. Wu Y, Lim J, Yang MH (2013) Online object tracking: a benchmark//IEEE Conference on Computer Vision & Pattern Recognition:2411–2418
11. Koide, Kenji & Miura, Jun. (2018). Convolutional Channel Features-Based Person Identification for Person Following Robots. doi:10.1007/978-3-030-01370-7_15.

APPENDIX

Mechanical Component List for Physical Robot:

- Chassis constructed from:
    - 2 x water-jet cut ⅛” thick aluminum 6061 panels (front and back)
    - 4 x ⅜” UNC, 19.5” long, stainless steel ready rod connecting front and back aluminum panels (panels spaced 15.5” apart)
    - aluminum angle bracket + 2” diameter caster wheel
- 2 x 8.5” (21.6 cm) diameter, 2-⅜” (6 cm) thick rubber wheels
- 2 x ½” diameter 44W steel wheel axles
- 4 x pillow block bearings for wheel axles
- 2 x couplers and insert pairs (coupling wheel axles to motor shafts)

Electrical Component List for Physical Robot:

- Battery (24VDC)
    - Turnigy, LiPo, 22.2V, 6S, 5200mAh, 10C constant discharge, 20C peak discharge
- Manual Motor Starter (MMS)
    - ABB MS116-6.3, 4.0-6.3A trip range (set at 4.2A = 2 x Motor Rated Current)
- Emergency Stop Button (E-Stop)
    - LAY37 Y090, 1 x SPST NO contacts, 1 x SPST NC contacts
- Motor Contactor (MC)
    - Songle Relay SLD-24VDC-1C, SPDT 30A/40A 14VDC contact rating, 50VDC max switching voltage, 40A max switching current
- 2 x Motor Drivers (Motor Driver Left & Motor Driver Right)
    - Cytron MD10C, 5-30VDC motor voltage range, 3.3V and 5V logic level input, 13A max continuous, 30A max peak for 10sec, 20kHz max PWM frequency
- 2 x Motors
    - IG42 TD-044-240, 24V DC, Brushed, 1:24 Gear Head, 8Kg-cm Rated Torque, 246RPM Rated Speed, 2.1A Rated Current, 13A Stall Current
- 2 x Encoders
    - 2 Channel, Hall Effect Magnetic, Incremental, Rotary, 5cpr without quadrature and 1:24 gear ratio, 480cpr with quadrature and 1:24 gear ratio
- Arduino Board
    - Arduino Mega 2560 Rev3
- Laptop
    - Dell, Intel i7-8550U CPU @ 1.80GHz, 16.0 GB RAM, Intel UHD Graphics 620
    - Linux Ubuntu 18.04 OS & ROS 1 Melodic running on ext. 500GB, USB3.0, SSD
- Camera
    - Intel RealSense D435i with IMU (accelerometer & gyroscope)
- Waypoint Indicator LED (blue)

Motor Power Wiring Diagram for Physical Robot

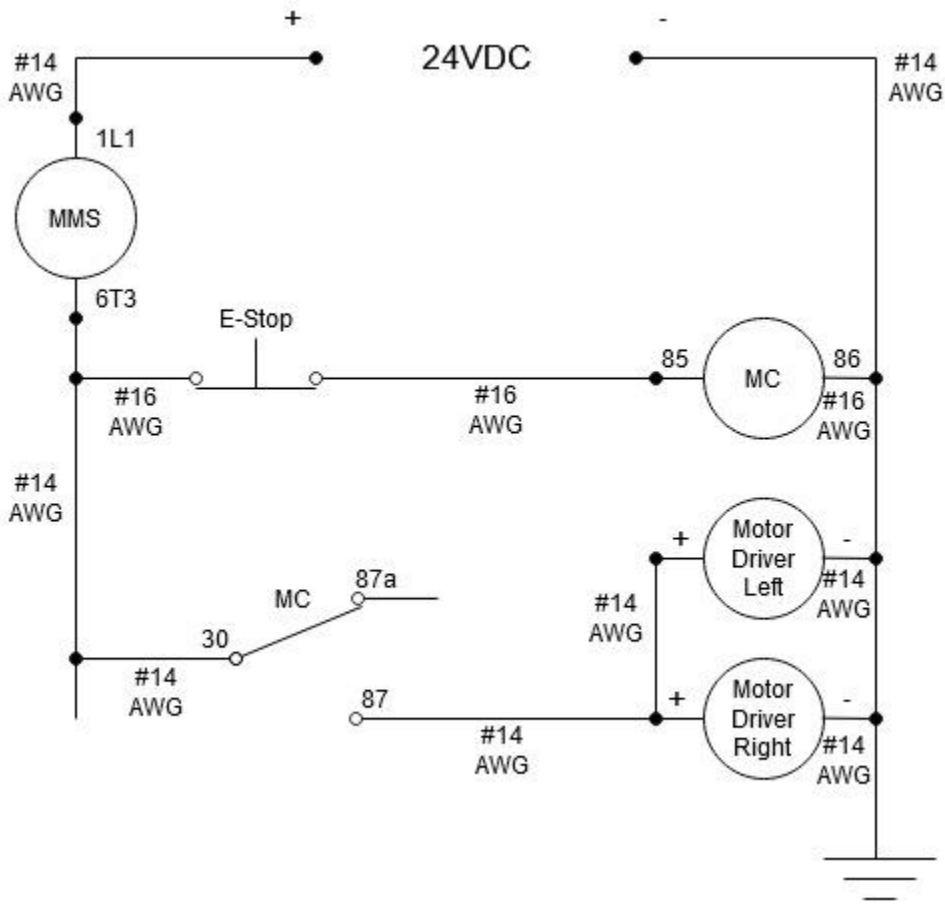


Figure 34: Motor Power Wiring Diagram for Physical Robot

Differential Drive Computation

The following equations are used by the differential_drive package to produce the odometry data, namely the **x-position**, **y-position**, and **yaw** of the robot's center of rotation relative to the fixed world frame:

$$D_{Left} = 2 \pi R_{Left} n_{Left} / N_{Left}$$
$$D_{Right} = 2 \pi R_{Right} n_{Right} / N_{Right}$$
$$D = D_{Right} + D_{Left} / 2$$

$$\Phi \mathrel{+}= D_{Right} - D_{Left} / L$$
$$x \mathrel{+}= D*\cos(\Phi)$$
$$y \mathrel{+}= D*\sin(\Phi)$$

Where
R_{Left} , R_{Right}*: radii of the left and right wheels, respectively, in metres*
L: wheel separation distance in metres
n_{Left} , n_{Right}*: encoder counts of the left and right wheel, respectively*

N_{Left} , N_{Right}: total encoder counts per revolution of the left and right wheels, respectively

D_{Left} : linear distance moved by left wheel in metres
D_{RIght} : linear distance moved by right wheel in metres
D: linear distance moved by robot's center of rotation in metres

***Φ:** yaw angular position in radians of robot relative to +x axis of fixed world frame*
***x**: x-position in metres of robot relative to fixed world frame*
***y**: y-position in metres of robot relative to fixed world frame*

Target Tracking Test on Carpet

Video : https://youtu.be/3xi6AV5KmyQ
Summary Description of Operation (also see Figure 13: Physical Robot State Machine):

- Fyi: Blue LED is:
    - ON when robot trying to obtain next waypoint
    - OFF when next waypoint obtained and robot traveling to this waypoint
- Please note: due to suboptimal PID values for wheel velocity control, robot sometimes takes a while to start rotating in desired direction
- Max velocity for either wheel is 0.25m/s

1. 03:16 in above video link: robot driving to current goal waypoint, blue waypoint indicator LED currently OFF (STATE4)
2. 03:23: robot reaches waypoint and stops, LED ON (STATE0)
3. 03:23: robot trying to keep target centered in view (3° angular deadband); if target not seen, robot rotates in direction it last saw the target (here CCW plan view) (STATE1)
4. 03:33: robot has seen target, target is within 2m distance limit and so rotates enough to keep target centered in view (STATE1)
5. 03:33-03:52: target still within 2m distance limit so robot continues to rotate to keep target centered (STATE1)
6. 03:52: robot loses sight of target and so rotates in direction it last saw target (CCW plan view) (STATE1)
7. 04:09: robot regains sight of target; target still within 2m distance limit (STATE1)
8. 04:12: robot loses sight of target and so rotates in direction it last saw target (CW plan view) (STATE1)
9. 04:26: robot regains sight of target, but target is now greater than 2m away so robot creates/obtains next waypoint; LED OFF (STATE2)
10. 04:26-04:28: robot turns to this new waypoint (STATE3)
11. 04:28: target within 3° angular deadband so robot starts driving straight toward waypoint (STATE4)
12. 04:41: robot reaches waypoint and stops, LED ON (STATE0)
13. REPEAT behaviour according to figure X: State Machine for Physical Robot

We found that the robot was able to adequately follow the target in ideal conditions (bright, consistent lighting and level, smooth terrain) according to its state machine. The following observations were made:

- Better wheel velocity PID values need to be found to ensure faster response to target wheel velocity values
- Slower target wheel velocities are better to minimize overshoot of linear and angular velocities until better turning and driving PID values are found. For the current laptop + arduino setup, slower target wheel velocities also allows for more accurate IMU readings, visual odometry localization via RTABMAP SLAM, and encoder counts and therefore odometry position
- The target tracking node needs to be further refined to avoid background clutter false positives of the target. The yellow ball minimized this compared to the red ball, but we still had to hide or remove some yellow objects in the environment shown in the video to avoid false positives.
- The algorithm that obtains the next waypoint needs to be improved as the robot sometimes doesn't drive straight enough (within 3 degrees angular deadband) to the next waypoint (see 06:38 in video)

PID Tuning Procedure for Physical Robot

1. Run the "differential_drive.launch" launch file
2. Run rqt_plot and subscribe to topics /lwheel_vtarget and /lwheel_vel published by the /lpid_velocity node; these represent the target left wheel velocity and the actual left wheel velocity, respectively
3. In a terminal, publish 0.25m/s as the /lwheel_vtarget value
4. In the above launch file, increase the Kp value until oscillations of the actual left wheel velocity in rqt_plot starts to occur; use ½ this Kp value
5. Increase Ki until the actual velocity reaches the target velocity in a reasonable time (oscillations will occur if Ki is too high)
6. Kd can be left at 0, but can be used to decrease the time it takes for the wheel to reach the target velocity after a disturbance
7. Repeat above for right wheel