IMPLEMENTATION OF A ROBOTIC CONVOY CONTROL USING GUIDANCE LAWS

A Major Qualifying Project Report Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for

the Degree of Bachelor of Science

in Robotics Engineering,

By Christopher Earley

Date: January 13, 2012

Approved:

_____

Professor Taskin Padir, Advisor

ABSTRACT

The goal of this project is to implement a semi-autonomous system consisting of two ground vehicles that simulate a convoy control scheme using operator control for the master and autonomous control for the slave. Using a control system based on platform kinematics in conjunction with the open source ROS framework, three different convoy scenarios are investigated using two Clearpath Husky A100 ground platforms and results are compared to computer simulation. The main contributions of this project are the development of a software framework for multi-vehicle convoys and the identification of vehicle kinematic model.

TABLE OF CONTENTS

TABLE OF FIGURES

1   INTRODUCTION

Over recent years a research trend has emerged that seeks to actively revolutionize mobility as we know it, research to develop autonomous ground vehicles. Numerous companies and agencies around the world are working towards developing automated solutions for a variety of applications that seek to benefit military and civilian interests. [1] In recent years the fruits of this labor have begun to work their way out of the lab and into active use. [2]

From automated vehicle parking to autonomous material convoying, the growing pool of applications seek to raise system efficiency and increase the capability of human operators by offloading trivial or tedious tasks to automated systems. This has the ability to transform a variety of resource mobility processes by lowering the level of active human participation needed for a task, allowing operators to focus on other responsibilities. [3] Additionally, by allowing autonomous systems to govern locomotion, the amount of error caused by human intervention or distraction will be reduced, thus lowering vehicle collisions while increasing overall efficiency. [4]

1.1   BACKGROUND

With the number of autonomous agents in active duty around the world rising every year, more and more applications are being explored in order to achieve a state where each human in the loop corresponds to multiple robotic agents in the field. This aspect of human-to-machine "force multiplication" has been a big focus of private and public research prompting the creation of multiple projects that seek to change everything from the daily commute to how resources will be mobilized in active war zones. [5] [6]

The sector that expects to be the training ground for some of these robotic agents is the military logistics subdivision, which in recent years has been strained to achieve the supply fulfillment requirements defined by the U.S. Army doctrine. [7] Current supply chains suffer from human unreliability and the growing need for continuous supplies is fast outpacing the ability for current personnel volumes to keep up. From this the need for some type of one-to-many system of force multiplication is a rising necessity.

One such system on the horizon is the Convoy Active Safety Technology (CAST) system being developed by Lockheed Martin for the US Army Tank Automotive Research, Development, and Engineering Center. CAST works as an installable kit for existing Army ground vehicles that integrate a host of sensors and actuators allowing the vehicle to be autonomously controlled. The outfitted vehicle can now act as a normal human-driven automobile or function autonomously as a slave convoy vehicle capable of following a lead unit though unknown terrain. [8]

Though the recent developments in robotic convoy technology are focused on more than military logistics applications, the EU-financed Safe Road Trains for the Environment (SARTRE) project aims to allow for normal motorists to join into an ad hoc convoy commanded by a professional driver. After joining the platoon, the vehicle will transition into an autonomous state and hold a set distance from the car before it as the unit progresses along the highway. The driver can stay in the platoon until the relevant exit is approached and then the car can disengage the platoon and transition back to human control for the remainder of the trip. This system, if successfully developed and adopted could revolutionize modern transport by lowering traffic fatalities from human error, increasing fuel economy, and freeing commuters to focus on things other than driving. This project is currently progressing from the simulation phase to real-world testing at the Volvo Proving Ground near Gothenburg, Sweden. [9]

1.1.1 ROBOTIC CONVOYS

At the core of this logistics issue lie the convoy, a simple but effective manner of moving equipment and supplies over long ground distances using multiple vehicles and a set path. This manner of assembly affords the unit many useful logistic features not limited to, increased efficiency due to the drafting effect of following a in a vehicle's wake, low logistic overhead since all traveling parties are using the same route, and high recoverability since the total supply package is split among the vehicles versus being contained in one vulnerable unit.

The problem of designing a force multiplying convoy system can be boiled down to a set of simple requirements:

1. The lead vehicle, driven by a human, must be functionally identical to a normal convoy lead vehicle meaning that all moment-by-moment decisions for the following units must be separated from the duties of the lead driver.

2. The following units must be able to follow the path set by the lead vehicle and be able to maintain a semi-static distance with the unit before them.

3. In addition, the following units must be able to deviate from the set path in the case of dynamic conditions on the set path. After executing the deviation, the units must be able to return to the path proper at the set distance.

This project focuses on the implementation of requirements 1 and 2 to develop a functional two platform convoy system using two Clearpath Husky A100 ground vehicles. This research will serve as the Major Qualifying Project[1] [MQP] for fulfillment of a bachelor's degree in Robotics Engineering at Worcester Polytechnic Institute.

In our preliminary research into convoy dynamics, numerous relevant papers were published covering a plethora of various solutions to the partial-autonomy convoy problem. [10] [11] [12] [13] Seeing as our focus for this research was to find a workable solution while reducing overall system complexity we focused on a paper written by Fethi and Boumediene Belkhouche that applied kinematic guidance laws to create a dynamic control system driven convoy. [14]

Whereas most convoy control solutions rely on control methods like visual servoing [15] [16] [17] or fuzzy logic systems [18] [19] [20] that made decisions from external sensor data, the methods outlined in this paper rely on control systems derived from kinematic models of holonomic ground platforms. This lowers the computational burden imposed on the system while allowing for constant control of slave units even if visual contact with the leader is lost. For this system to work, assumptions are made to ensure a proper convoy starting arrangement where all units are in known positions and distances from each other and that the units have free access to the live velocity and pose information regarding their leaders.

To control the two mobile platforms, a unified software framework was needed to connect each system in an asynchronous network capable of transferring sensor data while allowing for data collection and control of the platform's movement.

---

[1] MQP is a capstone project that culminates a student's WPI education with the application of ideas from their coursework to designing a product, instrument, device, or program which integrates theory and practice.

1.1.2 CONVOY VEHICLE SETUP

In this convoy, we have two ground platforms with known positions within a global coordinate frame. The platform positions are defined as (x_i, y_i) for the master unit and (x_{i+1}, y_{i+1}) for the following slave unit with orientation angles of θ_i and θ_{i+1} respectively. The angle of the direct line-of-sight between the vehicles is denoted as $\sigma_{i.i+1}$ and has a desired length of d_0.
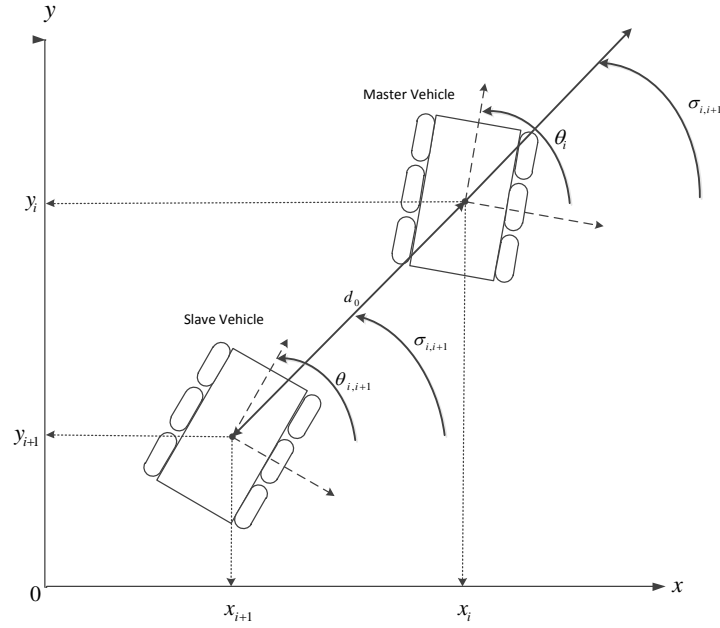
FIGURE 1: CONVOY SYSTEM SETUP

Before the convoy vehicles are started, the platforms must be separated by the desired follow distance with both vehicles facing each other. In this case we can now place the master vehicle at the origin of the global coordinate frame with the slave offset along the x axis at a distance of $-d_0$. This configuration simplifies the initial vehicle poses and makes the convoy problem an issue of maintaining the follow distance and slave pose with relation to the master.

x

Master Vehicle

y

0

d_0

Slave Vehicle

FIGURE 2: INITIAL POSITION AND ORIENTATION OF THE LEADER AND FOLLOWER IN CONVOY FORMATION

1.1.3 SOFTWARE FRAMEWORKS

The software development framework ROS or robotic operating system was chosen as our unifying platform due to its stability and existing support 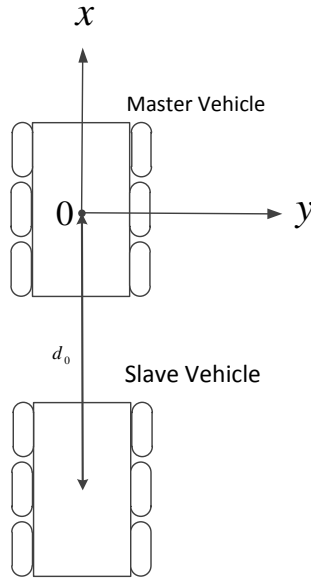for our ground vehicle. ROS is a multi-platform system aimed at creating singular computing clusters out of heterogeneous hardware units with a standard communication platform connecting the units together. This system began in 2007 at Stanford's Artificial Intelligence Lab for their mobile robotic platform named STAIR (Stanford Artificial Intelligence Robot). In 2008 the California robotics firm Willow Garage released the first version of their flavor of Switchyard they called ROS with a focus on extend the system to easily accept various hardware platforms, sensor packages, and the algorithms needed to tie them together into a working compliant system. Since its release in 2008, ROS had become a major player in the robotic development platform world with thousands of ROS framework elements, or packages, available and used by an active community of developers and companies.

1.1.4 MOBILE ROBOT PLATFORMS

To implement the convoy logic needed for autonomous control a set of hardware and accompanying software must be acquired or created. For this project we opted to utilize the Clearpath Husky A100 6-wheeled skid-steered

5

mobile platforms available for department research. These pre-built robotic systems not only afforded us a turn-key hardware solution but also a provided C++ library and official support for widely used robotic frameworks. Lastly, since this application implies the possibility of natural or rough terrain, the rugged water-resistant body of the Huskies would prove to be an additional benefit for outdoor environments. The Clearpath Robotics Husky A100 is a differential drive system built for rough terrain and large open spaces. It has a full micro ATX computer assembly on board along with a set of regulated voltage supplies for powering additional components that provides a nominal operation time of two hours and a peak power of 600 W.

FIGURE 3: 3D RENDER OF THE A100 PLATFORM

2 CONTROL BASED ON GUIDANCE LAWS

Based from the research into convoy pursuit methods, a strategy grounded on generating a set of control laws for the angular and linear momentum of the slave vehicle based of relative velocities was adopted and implemented. The control strategies used in this project were directly derived from guidance law formulations using the kinematic states as a platform for generating a mathematical formulation of their relative positions and velocities over time. Starting with the most basic case, the velocity pursuit, two other versions of guidance law will be discussed and tested.

2.1.1 VELOCITY PURSUIT

To maintain a convoy arrangement, control velocities that relate a slave and its master are calculated in the global coordinate frame. This is done using guidance law equations that take advantage of the available pose information of the platforms. First, the geometric relationship between the two platforms is defined and from that, control signals are generated to maintain a motion matching steady-state.
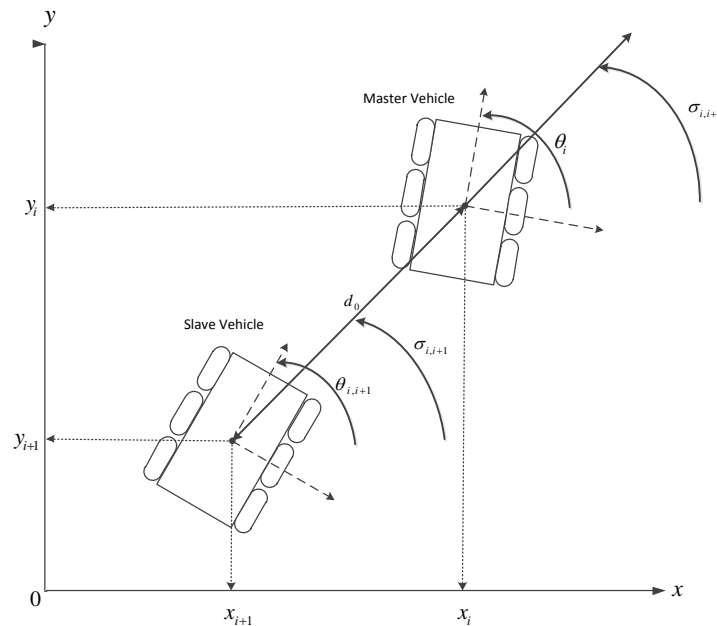

FIGURE 4: PLATFORM GEOMETRY

To relate the two platforms, the line of sight angle $\sigma_{i.i+1}$ and the relative unit distance $r_{i.i+1}$ are calculated using the planar coordinates of the two platforms denoted here as (x_i, y_i) for the master unit and (x_{i+1}, y_{i+1}) for the following slave unit.

$$\sigma_{i.i+1} = \tan\left(\frac{y_i - y_{i+1}}{x_i - x_{i+1}}\right), \ r_{i,i+1} = \sqrt{(y_i - y_{i+1})^2 + (x_i - x_{i+1})^2}$$

From those values we can now solve for the thrust velocity of the slave unit,

$$v_{i+1} = v_i + \cos(\theta_i - \sigma_{i.i+1})$$

where v_i is the scalar velocity of the lead unit and θ_i is the global orientation of the vehicle.

With this velocity known, it can now be separated into the individual vector components with respect to the line of sight angle and sent to the slave system. The equations below are used for the simplest version of the convoy guidance law systems, the velocity pursuit.

$$\dot{x}_{i+1} = v_{i+1} * \cos(\sigma_{i.i+1})$$
$$\dot{y}_{i+1} = v_{i+1} * \sin(\sigma_{i.i+1})$$
$$\dot{\theta}_{i+1} = \dot{\theta}_{i,i+1}$$

Now in the above example, only the angle and the velocities of the two systems are controlled with no attention spent on maintaining a set distance between the platforms. This can be rectified by adding the desired distance, d_0 to the slave velocity calculations as follows.

$$\dot{x}_{i+1} = v_{i+1} * \frac{(x_i - x_{i+1})}{d_0}$$

$$\dot{y}_{i+1} = v_{i+1} * \frac{(y_i - y_{i+1})}{d_0}$$

$$\dot{\theta}_{i+1} = \dot{\theta}_{i,i+1}$$

2.1.2 DEVIATED PURSUIT

An extension to the standard velocity pursuit algorithm, the deviated pursuit includes a small $(< \pm 10°)$ constant deviation angle α_{i+1} to the velocity calculation. The inclusion of this angle provides the following unit the ability to converge with the leader's path with greater accuracy then can be achieved with the standard velocity pursuit.

$$\dot{x}_{i+1} = v_{i+1} * \cos(\sigma_{i.i+1} + \alpha_{i+1})$$

$$\dot{y}_{i+1} = v_{i+1} * \sin(\sigma_{i.i+1} + \alpha_{i+1})$$

$$\dot{\theta}_{i+1} = \dot{\theta}_{i,i+1}$$

2.1.3   PROPORTIONAL PURSUIT

Another variation of pursuit law is the proportional pursuit which includes a small $(< 1 \pm 0.1)$ scaling constant K to the line of sight angle and angular velocity. This once again allows the following unit the ability to converge with the leader's path with greater accuracy than can be achieved with the standard velocity pursuit.

$$\dot{x}_{i+1} = v_{i+1} * \cos(K\sigma_{i.i+1})$$

$$\dot{y}_{i+1} = v_{i+1} * \sin(K\sigma_{i.i+1})$$

$$\dot{\theta}_{i+1} = K\dot{\theta}_{i,i+1}$$

The three versions of guidance law are each capable of generating the velocities needed to allow a slave vehicle to follow a master. Though, for the two special cases of velocity pursuit, the proper choice of constants could have an overall positive effect on the slave vehicle's ability to fully converge with the master's path. These results will be examined later though simulation in an attempt to reproduce the results shown in the original research.

3 SOFTWARE FRAMEWORK

Since we were to be using two husky A100 ground vehicles in concert to simulate the smallest convoy unit possible, a software platform that eased the development of the required inter-system communication was a must. In addition, since there was a large selection of dynamic data outlined in the paper that would be updated and broadcasted at any point in time, the ability to perform multi-threaded processes to react to the rapid changes in state was heavily favored. From these requirements two possible systems were discovered, first was straight C/C++ program development using the provided Clearpath libraries in conjunction with TCP/IP sockets, the second was the Robotic Operating System [ROS] framework maintained by Willow Garage.

After initial research into development with the two platforms, it proved to be that creating an asynchronous multi-threaded communication system in C++ would become one of the bigger drawbacks to developing with the provided libraries. This was greatly contrasted by the ease of multi-system integration that ROS was known for, seeing that it was design to easily create homogeneous computing clusters from heterogeneous systems over UDP/TCP.

Since the master unit will be human controlled, a method of user input was needed. Since the platform for ROS is a unix-based operating system, the use of USB videogame joy pads was made trivial. For ease of use, enhanced debugging, and data collection, the user control pad was connected to a mobile laptop that in turn connected to the wireless hub installed on the lead robotic platform. This setup prevented the driver from needed to be tethered to the mobile platform and allowed for a stationary interface to launch the software from or perform diagnostics. Lastly, the follower platform would also connect to the leader's wireless hub to complete the network graph.
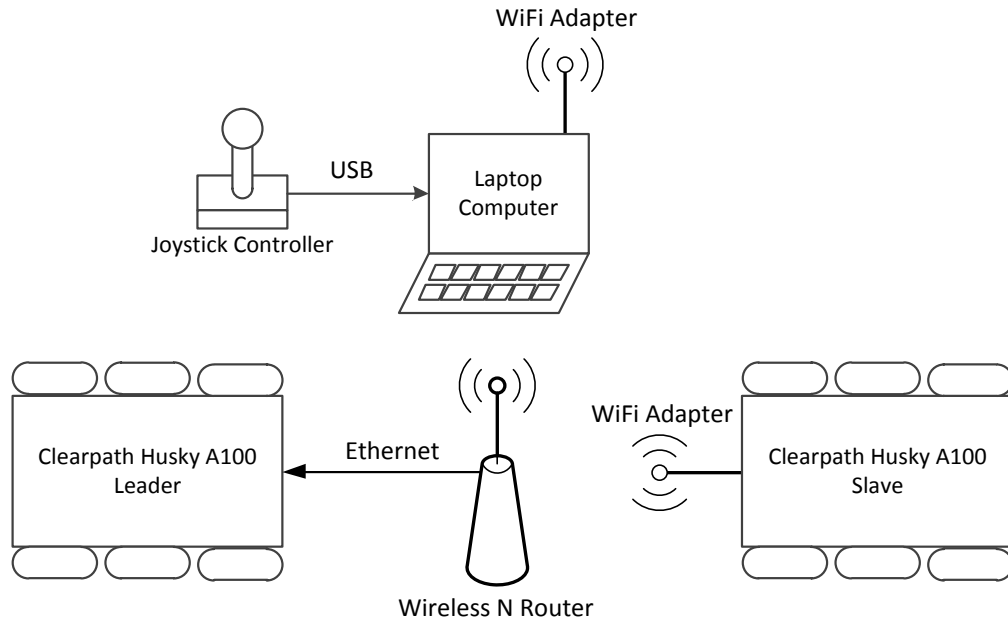
FIGURE 5: SYSTEM BLOCK DIAGRAM

Within each Clearpath Husky A100, there exists a set of quadrature encoders connected to the drive shafts of the wheel assemblies that connect the three wheels on each side. These encoders generate timing signals that are interpreted by the A100 firmware and converted to linear/angular velocities and travel distances for the platform in real time, in meters per second and meters respectively. Since the gearing and the radius of the wheel are static values that cannot be changed by the owner, the automatic conversion from angular wheel speed to linear velocity can be considered valid and usable for later calculations.

3.1.1 GUIDANCE LAWS

For each of the guidance law based convoy algorithms defined in the paper by Fethi and Boumediene Belkhouche, a ROS node was developed capable of executing the algorithm on live velocity and pose data published by the two clearpath platforms. The similarities in each convoy algorithm allowed for comparable design patterns in how each algorithm's node consumed data, performed the described operations, and finally published the resulting velocities.

In the code for each version of the convoy algorithm, the flow of each program is identical in structure. The general process is as follows:

1. Initialize variables and setup topic subscription and publishing

2. Update internal variables with the latest pose and velocity data from subscribed topics

3. Perform the guidance law calculations

4. Publish the resulting velocities for other nodes to consume

3.1.2 ROS System Development

Development in ROS revolves around the concept of a graph, where each node is a separate process and the edges connecting them are communication lines for sharing data. Each node is its own individual executable package that can be programmed in a variety of mainstream languages, including python and C++. For this system, we opted to use python due to its high level of support within the ROS community. Within these nodes, communication is setup using a publisher/subscriber model. In this nodes can send out data to the rest of the graph by creating a publisher on a unique topic with a set message structure.

An example of a publisher object would look like this:

```
cmd_pub = rospy.Publisher('cmd_vel', Twist)
```

Where "cmd_vel" is the label of the topic that is getting created and "Twist" is a defined message structure that consists of two, three float value arrays that represent the linear and angular velocities for a three dimensional object. After this object is created, it can then be used to send Twist data to an arbitrary number of other nodes. The code required to actually publish the data contained in a populated Twist message object is as follows.

```
cmd_pub.publish(self.twist)
```

When this command is executed, whatever data is within the self.twist object is sent out over TCP to a backend central ROS core server that then handles the rest of the communication to subscribing nodes.

Subscriptions work in a similar manner, during the initialization phase for a node the subscriber can be called that maps the data from a given topic, for example "/robot/cmd_vel" of the message type Twist, and upon reviewing a new message, run the function receivedTwist.

```
rospy.Subscriber("/robot/cmd_vel", Twist, self.receivedTwist)
```

Now every time new data is transmitted on that topic, the service routine method receivedTwist will be executed and in most cases the called method will just copy the new information to an object for use in other parts of the program.

With the custom equations now wrapped into executable ROS nodes that run continuously and publish data, the various stacks can be strung together to form the system capable of the functionality we outlined previously. This process is performed in ROS by creating a launch file that contains the information needed to execute each individual node and how they connect together to form the overall application structure.

The convoy application is split up into three hardware devices, the control unit, the slave platform, and its master. Each of these systems will run a subset of the application's nodes and during operation will pass information to each other asynchronously.
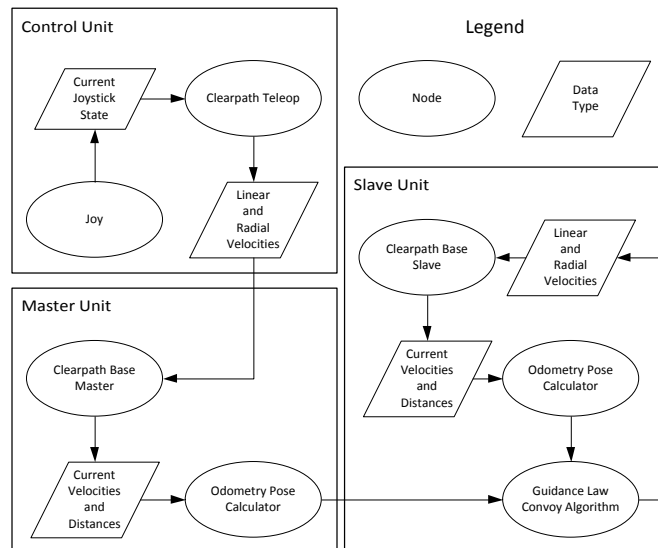

FIGURE 6: ROS NODE OVERVIEW

For the sake of reducing the TCP communication lag, nodes were made to run on systems that generated data that the node required for its calculations. That way the only time that a non-localhost data transfer will occur is when a sub-system is broadcasting data concerning its individual state to another set of nodes for further calculations.

For each of the clearpath platforms, a set of nodes for calculating pose estimates and turning velocity messages into platform movement is hosted locally. The slave unit has, in addition to the previously mentioned, a node for consuming current the poses and velocities of the two units and generating a velocity message in accordance with the guidance law algorithm being executed. Lastly, the control unit, acts as an interface USB joystick and converter for turning joystick state messages into velocities to be consumed by the master unit.

This structure was described in the launch file through the use of namespaces defined in the file's XML syntax. An example of a node being defined to run with a specific namespace is given below.

```xml
<!-- Setup nodes for lead robot wolfgang -->

    <group ns= "clearpath/robots/master">

        <node pkg="joy" type="joy_node" name="joy">

        </node>

    </group>
```

In this example, the controller messages generated from the joystick node will be broadcasted from the namespace "/clearpath/robots/master/joy". Any other node that needed this information was written to subscribe to that address specifically and upon successful subscription, will be able to receive live data regardless of the hardware platform the node is run on.

Once each node has been defined and given a namespace, the launch file can then be executed automating the process of executing the individual nodes on the appropriate hardware and setting up the needed communication framework. After the startup process, the system is ready to operate.

4 KINEMATIC MODEL IDENTIFICATION

In order to fully control a ground vehicle in situations such as convoys, the kinematic characteristics of the platform must be investigated to develop a kinematic model. In the case of this project, analysis was performed to calculate the movement constraints on the differential drive vehicle to gain a general understanding of the platform's limitations. Then a general model was created that allowed for the calculation of vehicle poses from the available wheel encoders. Lastly, the efficacy of this model was tested on multiple surfaces to see how the vehicle constraints influence pose calculation.

4.1.1 PLATFORM CONSTRAINTS

Due to the differential wheel layout of our robotic platform, constraints were bound to exist in the system that limits the controllable degrees of freedom that we can use for kinematic control. To illustrate these restrictions, a nonholonomic sliding and rolling constraint analysis was performed on the A100 wheel geometry to derive a set of equations that express the movement concerns for each wheel and how they contribute to the entire system.
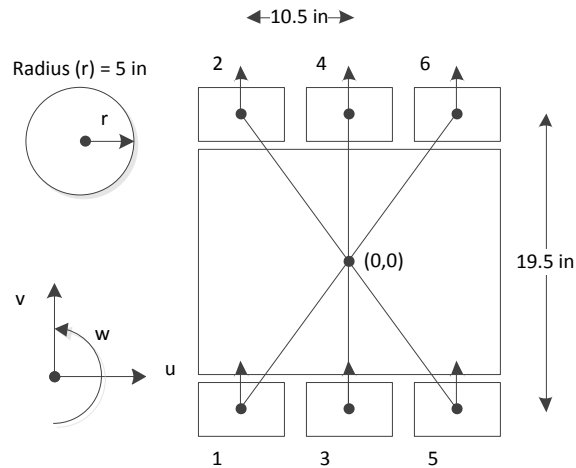

FIGURE 7: CHASSIS LAYOUT

After taking measurements of the A100 chassis' width and wheel distances, a diagram of the system's layout was generated and the relevant metrics computed. In order to calculate the constraints of the system, there are 3 data points that must be found for each wheel.

1. L : The distance of the center of the wheel from the geometric center of the platform, marked as the origin in the diagram.

2. α : The angle of the wheel position with respect to the robot coordinate frame

3. β : The angle of the right-hand-rule derived direction needed for the wheel to go forward orientation with respect to the angle α

Wheel	1	2	3	4	5	6
L (inches)	14.33	14.33	9.75	9.75	14.33	14.33
α (degrees)	-137.12	137.12	-90	90	-42.88	42.88
β (degrees)	-137.12	-47.12	-180	0	137.12	47.12

With these values calculated we can now begin to generate equations to describe the effect the motion of the platform's wheels will have on the platform velocities expressed in the global coordinate system, $\dot{q}_0$. The platform velocities are represented by the linear velocities u and v and the angular velocity w. Since there are constraints on the robot's motion, we have to convert the vectors from the global coordinate frame to a frame related to the platform. This process is accomplished through the use of the rotation matrix, R^T that relates vectors from a global coordinate system to the local coordinate system used by the platform.

$$\dot{q}_0 = R \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

$$R^T = \begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

It must be said that in these constraint models, we assume that the wheels are incapable of slipping or deforming, that there is no friction on the rotational axis, and that the steering axes are normal to the surface.

4.1.2   Rolling Constraints

The set of equations below seek to characterize in what direction the wheels will begin to roll if acted upon by an external force.

16

$$[\sin(\alpha + \beta), -\cos(\alpha + \beta), l\cos(\beta)]R^T\dot{q}_0 = r\dot{\varphi}$$

$$\text{Wheel 1: } [0.997, -0.074, 10.5]R^T\dot{q}_0 = r\dot{\varphi}$$

$$\text{Wheel 2: } [1, 0. -9.75]R^T\dot{q}_0 = r\dot{\varphi}$$

$$\text{Wheel 3: } [1, 0, 9.75]R^T\dot{q}_0 = r\dot{\varphi}$$

$$\text{Wheel 4: } [-1, 0, -9.75]R^T\dot{q}_0 = r\dot{\varphi}$$

$$\text{Wheel 5: } [0.997, 0.074, 10.5]R^T\dot{q}_0 = r\dot{\varphi}$$

$$\text{Wheel 6: } [1, 0, -9.75]R^T\dot{q}_0 = r\dot{\varphi}$$

The zero or near zero values present in the constraint values for each wheel point to our platform being unable to roll in all degrees of motion afforded by the two dimensional plane. This means that this system, for most intents and purposes, can only move along the direction of the surge velocity, u without having to undergo slipping.

4.1.3 SLIDING CONSTRAINTS

These constraints represent in which direction the wheels will begin to slip when acted upon by an external force, such as a force generated by the other wheels.

$$[\cos(\alpha + \beta), \sin(\alpha + \beta), l\sin(\beta)]R^T\dot{q}_0 = 0$$

$$\text{Wheel 1: } [0.074, 0.997, -9.75]R^T\dot{q}_0 = 0$$

$$\text{Wheel 2: } [0, 1, -10.5]R^T\dot{q}_0 = 0$$

$$\text{Wheel 3: } [0, 1, 0]R^T\dot{q}_0 = 0$$

$$\text{Wheel 4: } [0, 1, 0]R^T\dot{q}_0 = 0$$

$$\text{Wheel 5: } [-0.074, 0.997, 9.75]R^T\dot{q}_0 = 0$$

$$\text{Wheel 6: } [0, 1, 10.5]R^T \dot{q}_0 = 0$$

The zero or near zero values present in the constraint values for each wheel point to our platform being unable to slide in all the degrees of motion afforded by the two dimensional plane. Additionally, if any considerable force is applied along the orthogonal axis, slippage will occur.

Although this is common knowledge for anyone that has used a skid-steer platform, this exercise proves that this platform is indeed non-holonomic since there exists a discrepancy between the total degrees of freedom expressed in the vector $\dot{q}_0$ and the total controllable degrees of freedom expressed by the constraints.

4.1.4 KINEMATIC MODEL

To use the platform for this project we needed to generate a kinematic model that would allow us to calculate the system's state relative to a global coordinate frame using the on-board quadrature encoders. Since using the encoder to generate a location and orientation estimate over an extended period of time would accumulate a large amount of error due to slip, we used a discretized version of an odometry pose model that would update the system pose every time step by looking at only the change in distance and angle that occurred during the small time increment.

Beginning with the wheel distances calculated by the A100 firmware, the change in distance from the previous time step to the current is calculated for each side of the platform.

$$\Delta d_r = d_{r\,(t)} - d_{r\,(t-1)}, \qquad \Delta d_l = d_{l\,(t)} - d_{l\,(t-1)}$$

From that we can calculate the change in angle and linear distance for the entire platform where w is the width of the platform.

$$\Delta\theta = \frac{\Delta d_r - \Delta d_l}{w}, \quad \Delta d = \frac{\Delta d_r + \Delta d_l}{2}$$

Finally, the generated deltas are then summed with the previous position and angle values after calculating their respective trigonometric components.

$$x_t = x_{t-1} + \Delta d \cos(\theta_{t-1} + \frac{\Delta\theta}{2})$$

$$y_t = y_{t-1} + \Delta d \sin(\theta_{t-1} + \frac{\Delta\theta}{2})$$

$$\theta_t = \theta_{t-1} + \Delta\theta$$

This set of equations describes the position and orientation of the mobile platform in the global coordinate frame.

4.2 ODOMETRY VIABILITY TESTING

One issue with this odometry pose system is that it does not account for the slip that results from the platform's constraints. In order to quantify the amount of error that results from this, a simple test was performed that compared the actual rotated angle to the calculated angle on two types on flooring, short pile carpet and linoleum tile.

FIGURE 8: ANGLE TESTING SETUP - INITIAL POSE AND POSE AFTER 45 DEGREE ROTATION

As shown from figure 7, a tape marker was placed on the floor of test space with the vehicle placed above it. With the vehicle facing parallel to one of the tape lines, a game controller was used to make the platform spin till it was parallel to one of the four desired angles provided by the lines (45, 90, 135, 180 degrees respectively) and the angle calculated from the ROS odometry node was recorded for later analysis. This process was repeated three times for each of the angle markers on both types of flooring.

5 Results

Using the control system based on platform kinematics in conjunction with MATLAB, the three different convoy scenarios were investigated with the aim to make a comparison to the computer models offered from the original research. Finally, the developed odometry model was tested with the intent of investigating the effect of the platform's kinematic constraints on calculating accurate vehicle poses. This process was performed with the Husky A100 platform on a set of surfaces with differing amounts of wheel friction to see if the resulting error was systematic or a function of the dynamic wheel grip.

5.1.1 Convoy Simulation

To prove the validity of the guidance law concept, a simulation script was developed to test the theoretical concept of the various algorithms with a simplified set of holonomic platforms traveling on a simple path. The guidance law implementation used for this simulation is an exact copy of the version used in the final ROS package.

A demonstration of the slave vehicle's ability to converge with a circular master path was created to see if the slave was capable of following the master on a curved path using the velocity pursuit guidance law.
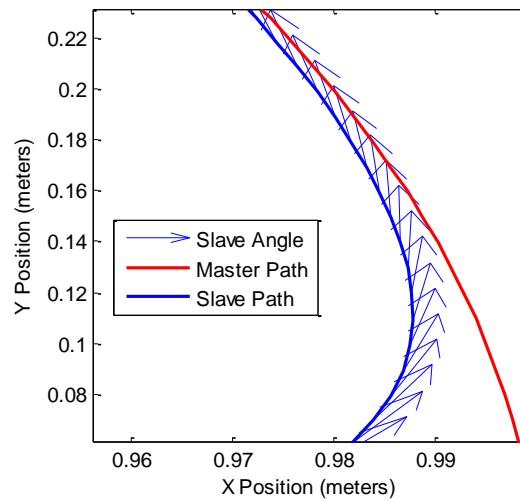

FIGURE 9: VELOCITY SLAVE PATH CONVERGENCE

Then a simulation utilizing the velocity pursuit with desired follow distance of 0.1 meters was made to test that the guidance equations would allow the slave to maintain the desired distance around a circular path.
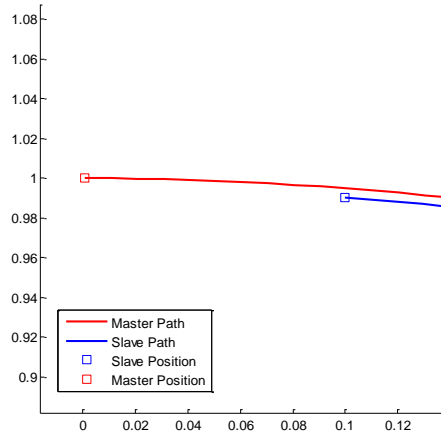
FIGURE 10: PURSUIT WITH DISTANCE OF 0.1 METERS

Finally, the three versions of guidance law were simulated on an identical circular path to compare their ability to match the movement of the master vehicle.
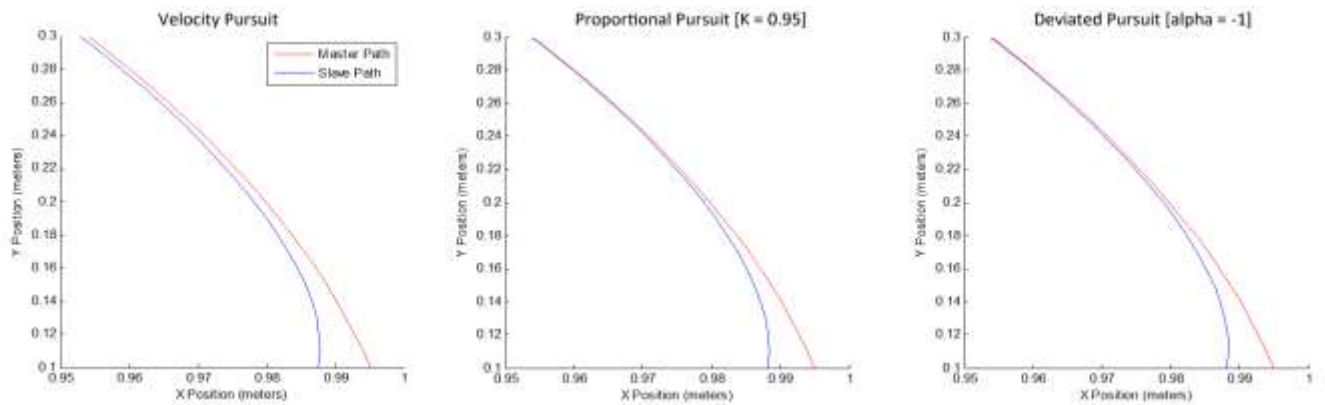

FIGURE 11: SLAVE PATH CONVERGENCE, COMPARISON OF CONVOY ALGORITHMS

5.1.2 PLATFORM POSE ESTIMATION ERROR QUANTIFICATION

Using tape markers placed on the testing surface, a Husky A100 ground vehicle was driven to four different angles, the resulting calculated pose angles from the odometry system were recorded and compared to the actual amount the vehicle rotated.
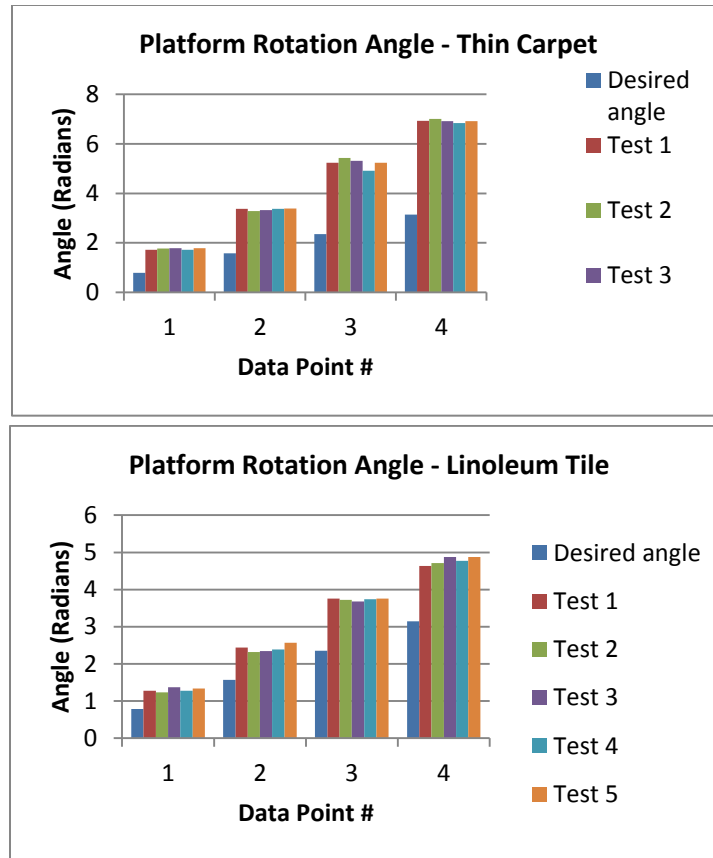
21

Platform Rotation Angle - Thin Carpet

Platform Rotation Angle - Linoleum Tile

FIGURE 12: ODOMETRY ANGLE CALCULATION - THREE TRIALS OF FOUR DESIRED ANGLES

Using the data from the four trials the average of the recorded angles were computed and plotted against the desired angles that the vehicle actually traveled.
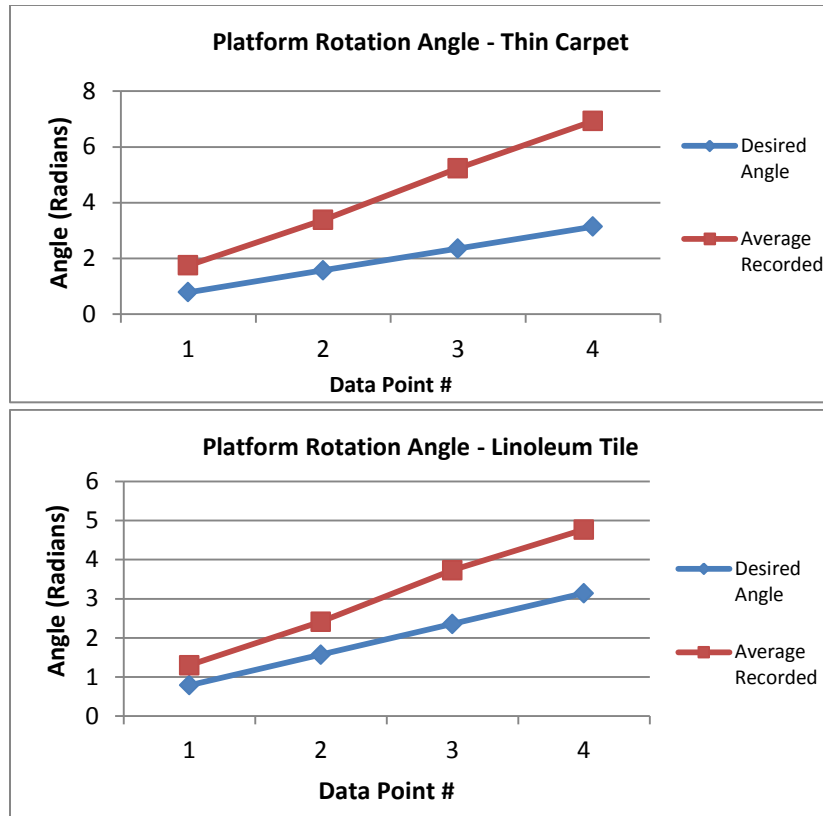
FIGURE 13: COMPARISON OF KNOWN ROTATION ANGLES AGAINST ODOMETRY CALCULATED POSE ANGLES

With the average reported angles known, the angle errors for both surfaces were computed and compared to one another to illustrate the variation of odometry error caused by surface friction during turns.
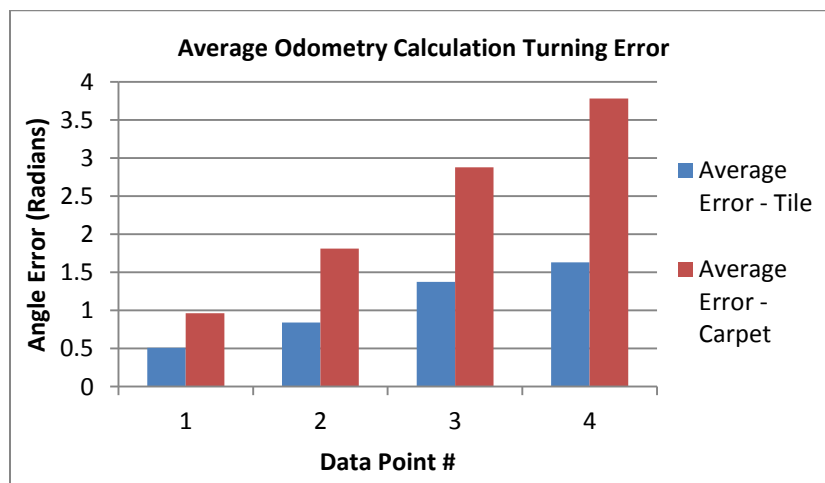

FIGURE 14: AVERAGE ERROR OF ODOMETRY ANGLE CALCULATIONS FOR FOUR DESIRED ANGLES ON TWO SURFACES

6 Discussion and Future Work

The ability of the velocity pursuit algorithm as described in the original paper was successfully recreated in simulation. As shown in the results, we were able to develop a scenario where the slave platform converged onto the master's path after starting a set distance away using velocity pursuit. In the case of the velocity pursuit variation that specified a mandatory following distance, the slave platform was able to stay away from the master at the correct distance for the entirety of the path. Lastly, the converging characteristics of the three variations of guidance law were simulated and the behaviors exhibited were similar to the original research. Both proportional and deviated pursuit generated accurate paths for the slave vehicle with greater fidelity then normal velocity pursuit.

Due to the aforementioned inaccuracies caused by the excessive skid combined with odometry derived pose calculation, the actual physical implementation of the guidance law convoy was not able to be accomplished in the time frame of the project with the tools available. Since the calculated poses of the two platforms were inherently faulty and dependent on the variable traction provided by the driving surface, any later calculation based on these values would exhibit behavior not defined by the actual state of the world. This result itself stresses the necessity of improvements to the pose estimation methods. Implementing such changes could rectify this issue and serve as a good starting place for any individuals wanting to continue development of this system.

6.1.1 Recommendations for Future Development

The work performed here can be considered preliminary groundwork in developing the scaffolding for a two unit convoy system with the given hardware platforms. Although much work has been done to build the framework needed to compute the velocities needed for the slave unit to follow its master, the actual tests show that due to the large error generated during odometry pose calculations, the resultant slave movement profiles perform erratically. This issue most likely stems from the slip-dependent dynamics of the six-wheeled chassis causing large deviations in the pose estimate derived from the encoder data, which does not take into account the generous amount of slip needed to perform a turn.

Additional sensors for calculating poses would be beneficial addition to the system. A combination of GPS and IMU would go great lengths to give decent pose estimates in a global coordinate frame. Within the ROS package repository, there exist multiple stacks that could be used to extend the existing framework to integrate the new data sources into the existing node graph. One possible combination would be to combine data from a commercial USB GPS dongle with a Nintendo Wiimote using a filtering algorithm to generate a unified pose. This process would be performed on both A100 platforms and used as the basis for any calculations regarding the positions of the two systems in the global frame.

Lastly, with dependable position data available, the system should be extended to allow for the slave unit to deviate from the control law dictated path to avoid obstacles. This deviation should occur when the possibility of collision with an external object becomes significant. After the obstacle is cleared, the system should revert back to normal control law convoy tracking. Fuzzy logic or probability based approach could be used as possible methods of controlling this behavior swapping that occurs when faced by an obstacle.

7   APPENDICES

7.1   APPENDIX A: ROS INTRODUCTION PRESENTATION SLIDES

1/31/2012

ROS packages - Description

- A package [or node] is the functional atom of a ROS application
- Communicates using a TCP/IP based broadcast & subscription message system
- Can be coded in a variety of languages.
  - Though C++ and Python are the most supported
- Can also create services that can be invoked by multiple executing nodes

Worcester Polytechnic Institute

ROS Topics and Messages

- ROS nodes generate messages that are broadcast on topics that are defined in a directory hierarchy format
  - /clearpath/robots/polly/pose
- Any other node can subscribe to a published topic as long as they know where to look
- Any topic can also be remapped or augmented through the remap tag upon launch-time

Worcester Polytechnic Institute

ROS Packages - Structure

ROS Stack File Structure

- Creation of new stacks/packages is easy with a prebuilt roscreate command
- Most of what is shown is automatically generated
- Information concerning each package/stack is stored in XML files located in the appropriate directory level
- Each package directory will contain source code, binary executables, testing sets, etc

Worcester Polytechnic Institute

ROS Launch – Tying it all together

- A system of nodes are executed on their specified host machine using a .launch file
- Each launch file contains
  - Information describing all the machines connected in the system and how to contact them over SSH/TCP-IP
  - Information needed to run a specified ROS node on a connected machine

Worcester Polytechnic Institute

ROS Application – Launch Structure

```
<launch>
  <machine ></machine>
    <group ns= "root/dir1/dir2">
      <node> <param/> </node>
    </group>
</launch>
```

Worcester Polytechnic Institute

Sample ROS Node Graph

Worcester Polytechnic Institute

WPI   Question Break

Question Break

Worcester Polytechnic Institute

WPI   ROS Example – Convoy MQP

- ROS is currently being used to develop a two-unit partially autonomous convoy.
- Currently working on implementing the convoy guidance laws described by Fethi and Bournediene Belkhouche
- Utilizes 3 separate hardware platforms all communicating using ROS to share information

Worcester Polytechnic Institute

WPI   Convoy MQP – Functional Overview

- Project goals
  - Have a user-controlled leader system operated via teleop interface
  - Have an autonomous slave system follow the leader while maintaining a set follow-distance
- Future Goals
  - Fully integrate sensors to allow for slave object avoidance

Worcester Polytechnic Institute

WPI   Convoy MQP - Process Graph
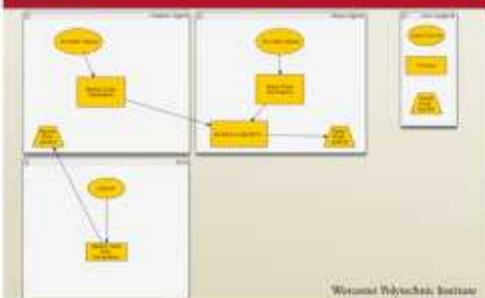
Worcester Polytechnic Institute

WPI   Convoy MQP – Node List

- Polly [Master]
  - Husky Hardware Interface
  - Pose Estimation
- Wolfgang [Slave]
  - Husky Hardware Interface
  - Pose Estimation
  - Guidance Algorithm Node
- Sledge [Controller]
  - Joy Stick Interface
  - Joy Value to Polly Drive Velocity Message Converter

Worcester Polytechnic Institute

WPI   Convoy MQP – Reasons for ROS

- ROS allowed for:
  - Easy communication between systems
  - Quick turn-key hardware support
  - Large documentation wiki + tutorials
- ROS issues were:
  - Steep learning curve
  - Lacking helpful build/run-time error messages + debugging
  - Hard to find packages

Worcester Polytechnic Institute

7.2 APPENDIX B: ROS NODE CODE

```python
#!/usr/bin/env python

# velocity_pursuit.py
# This is the ROS node that performs the velocity pursuit slave movement
# calculations. This node consumes live pose information from the two husky
A100
# systems and generates a twist velocity message for the slave unit to
consume
# Author: Chris Earley

import roslib; roslib.load_manifest('convoy_follow')
import sys
import rospy
import math
from convoy_follow.srv import *
from geometry_msgs.msg import Twist
from geometry_msgs.msg import PoseStamped
from convoy_follow.msg import PoseStampedVel
from clearpath_base.msg import *

class Velocity_Pursuit:

    def __init__(self):
        rospy.init_node("velocity_pursuit")

        self.turnscale = 5
        self.twist = None
        self.followPose = PoseStampedVel()
        self.leadPose = PoseStampedVel()
        cmd_pub = rospy.Publisher('cmd_vel', Twist)
        rate = rospy.Rate(rospy.get_param('~hz', 10))

        # setup calculated pose subscriptions
        rospy.Subscriber("/clearpath/robots/wolfgang/pose",    PoseStampedVel,
self.setFollowPose)
        rospy.Subscriber("/clearpath/robots/polly/pose",       PoseStampedVel,
self.setLeadPose)

        # calculate twist for slave at 20Hz
        while not rospy.is_shutdown():
            self.callback()
            # publish calculated twist message
            cmd_pub.publish(self.twist)
            rate.sleep()

    # calculated pose ISRs
    # update pose data after recieving new information
    def setFollowPose(self, data):
```

```python
        self.followPose = data
        print "updated FollowPose!"

    def setLeadPose(self, data):
        self.leadPose = data
        print "updated LeadPose!"

    def callback(self):
        print "Calculating Velocity Pursuit!"

        sigma_i_ip1    =    math.atan2(self.leadPose.pose.pose.position.x    -
self.followPose.pose.pose.position.x,    self.leadPose.pose.pose.position.y    -
self.followPose.pose.pose.position.y)

        r_i_ip1                                                               =
math.sqrt(math.pow(math.fabs(self.leadPose.pose.pose.position.x              -
self.followPose.pose.pose.position.x),                 2)                     +
math.pow(math.fabs(self.leadPose.pose.pose.position.y                        -
self.followPose.pose.pose.position.y), 2))

        if r_i_ip1 == 0:
            sigma_vel_i_ip1 = 999
        else:
            sigma_vel_i_ip1           =           self.leadPose.vel           *
math.sin(self.leadPose.pose.pose.orientation.z - sigma_i_ip1) / r_i_ip1

        vel_ip1                =                self.leadPose.vel              *
math.cos(self.leadPose.pose.pose.orientation.z                               -
self.followPose.pose.pose.orientation.z)

        x_vel = vel_ip1 * math.cos(sigma_i_ip1)
        y_vel = vel_ip1 * math.sin(sigma_i_ip1)
        theta_vel = sigma_vel_i_ip1 * 10

        # create a new twist message
        twist = Twist()

        # convert the global velocities to the robot's orientation frame
        twist.linear.x                =                x_vel                   *
math.cos(self.followPose.pose.pose.orientation.z)        +        y_vel        *
math.sin(self.followPose.pose.pose.orientation.z)

        twist.linear.y                =                x_vel                   *
math.sin(self.followPose.pose.pose.orientation.z)        +        y_vel        *
math.cos(self.followPose.pose.pose.orientation.z)

        twist.angular.z = theta_vel

        self.twist = twist
```

```python
if __name__ == "__main__":
    Velocity_Pursuit()
```

```python
#!/usr/bin/env python

# calculate_pose.py
# This ROS node consumes the live odometry information provided
# by the clearpath base node and calculates an estimated 2d pose
# for the system.
# As a warning, this node should not be used due to the large error
# that accumulates when the platform turns due to sliding.
# Author: Chris Earley

import roslib; roslib.load_manifest('convoy_follow')
import sys
import rospy
from convoy_follow.msg import PoseStampedVel
from geometry_msgs.msg import PoseStamped
from clearpath_base.msg import *
import math

class Calculate_pose:

    def __init__(self):
        rospy.init_node('calculate_pose')
        # set the width of the robot base
        self.base_width = 0.5 # 19 3/4 inches

        #init previous distances for delta encoder computation
        self.prev_dist_l = 0
        self.prev_dist_r = 0

        pose_pub = rospy.Publisher('pose', PoseStampedVel)
        # this published stamped pose was for some early rviz testing
    #   rviz_pose_pub = rospy.Publisher('rviz_pose', PoseStamped)

        self.pose = PoseStampedVel()
        # this value is the starting distance that the
        # robot is in at start
        self.pose.pose.pose.position.x = rospy.get_param('~init_pose_offset',
0)
        # setup subscription with topic encoders from the clearpath base node
        rospy.Subscriber("data/encoders", Encoders, self.callback)
        rate = rospy.Rate(20)

        while not rospy.is_shutdown():
            if self.pose:
```

```python
                pose_pub.publish(self.pose)
            #   rviz_pose_pub.publish(self.pose.pose)
            rate.sleep()

    def callback(self, data):

        # extract the encoder array from the encoders msg
        distance_left = data.encoders[0].travel
        distance_right = data.encoders[1].travel
        velocity_left = data.encoders[0].speed
        velocity_right = data.encoders[1].speed
            # calculate the movement deltas for the current time step
            delta_dist_l = distance_left - self.prev_dist_l
            delta_dist_r = distance_right - self.prev_dist_r
            # compute the current overall distance and angle for the robot
            delta_robot_distance = (delta_dist_l + delta_dist_r) / 2
            delta_robot_angle = (delta_dist_r - delta_dist_l) / self.base_width

            # create new empty pose vel object
            new_pose = PoseStampedVel()

        # populate the respose with the calculated pose values
        new_pose.pose.pose.position.x   =    self.pose.pose.pose.position.x   +
delta_robot_distance    *    math.cos(self.pose.pose.pose.orientation.z    +
delta_robot_angle/2)
        new_pose.pose.pose.position.y   =    self.pose.pose.pose.position.y   +
delta_robot_distance    *    math.sin(self.pose.pose.pose.orientation.z    +
delta_robot_angle/2)
        new_pose.pose.pose.orientation.z = self.pose.pose.pose.orientation.z +
delta_robot_angle

        new_pose.vel = (data.encoders[0].speed + data.encoders[1].speed)/2

            # update the self pose object
            self.pose = new_pose

            # update the prev distances at the end of the calculation
            self.prev_dist_l = distance_left
            self.prev_dist_r = distance_right

if __name__ == "__main__": Calculate_pose()
```

7.3 APPENDIX C: MATLAB SIMULATION CODE

```matlab
% velocityPursuit.m
% This script simulates a holonomic ground platform moving along a
% circular path with a slave unit following under the control of
% the Velocity pursuit guidance laws.
% The a subset of the paths for both system is printed to the
% screen as a figure.

clear; clc;
% Independent variable - time
tstep = 0.01;
t = 0:tstep:pi/4;
dio = 4;

% Desired x, y , th, u and w
xd_m = cos(t);
yd_m = sin(t);

dxd_m = diff(xd_m);
dyd_m = diff(yd_m);
thd_m = atan2(dyd_m, dxd_m);

ud_m = (dxd_m.^2 + dyd_m.^2).^0.5;
wd_m = diff(thd_m);

[m,n]=size(xd_m);

% init the previous values for the first calculation
xPrev_s = 0.9;
yPrev_s = 0;
thetaPrev_s = 0;

% calculate the velocities for the slave unit and simulate the movement
for i = 1:1:(n-1),
    sigma_i_ip1 = atan2((yd_m(i) - yPrev_s) , (xd_m(i) - xPrev_s));

    r_i_ip1 = ((yd_m(i) - yPrev_s).^2 + (xd_m(i) - xPrev_s).^2).^0.5;

    vFollow = 1;
    dxd_s(i) = vFollow * cos(sigma_i_ip1);
    dyd_s(i) = vFollow * sin(sigma_i_ip1);

    wd_s(i) = ud_m(i) * sin(thd_m(i) - sigma_i_ip1) / r_i_ip1 ;

    thd_s(i) = wd_s(i) + thetaPrev_s;

    xd_s(i) =  dxd_s(i) * tstep + xPrev_s;
    yd_s(i) =  dyd_s(i) * tstep + yPrev_s;
```

```
    xPrev_s = xd_s(i);
    yPrev_s = yd_s(i);
    thetaPrev_s = thd_s(i);
end

% Print out paths
%quiver(xd_s,yd_s,cos(thd_s),sin(thd_s),0.4);
hold on
plot(xd_m, yd_m, 'r');
hold on
plot(xd_s, yd_s, 'b');
box('on');
hold('all');

% Create xlabel
xlabel({'X Position (meters)'});

% Create ylabel
ylabel({'Y Position (meters)'});
```

```
% velocityPursuit_withDistance.m
% This script simulates a holonomic ground platform moving along a
% circular path with a slave unit following under the control of
% the Velocity pursuit guidance laws with a set follow distance.
% The a subset of the paths for both system is printed to the
% screen as a figure.

clear; clc;
% Independent variable - time
tstep = 0.01;
t = 0:tstep:pi/2;
dio = 0.1;

% Desired x, y, theta, u and w
xd_m = cos(t);
yd_m = sin(t);

dxd_m = diff(xd_m);
dyd_m = diff(yd_m);
thd_m = atan2(dyd_m, dxd_m);

ud_m = (dxd_m.^2 + dyd_m.^2).^0.5;
wd_m = diff(thd_m);

[m,n]=size(xd_m);

% init the previous values for the first calculation
```

```
xPrev_s = 1.1;
yPrev_s = 0;
thetaPrev_s = 0;

% calculate the velocities for the slave unit and simulate the movement
for i = 1:1:(n-1),

    sigma_i_ip1 = atan2((yd_m(i) - yPrev_s) , (xd_m(i) - xPrev_s));

    r_i_ip1 = ((yd_m(i) - yPrev_s).^2 + (xd_m(i) - xPrev_s).^2).^0.5;

    vFollow = 1;

    dxd_s(i) = vFollow * (xd_m(i) - xPrev_s)/dio;
    dyd_s(i) = vFollow * (yd_m(i) - yPrev_s)/dio;

    wd_s(i) = ud_m(i) * sin(thd_m(i) - sigma_i_ip1) / r_i_ip1 ;

    thd_s(i) = wd_s(i) + thetaPrev_s;

    xd_s(i) =  dxd_s(i) * tstep + xPrev_s;
    yd_s(i) =  dyd_s(i) * tstep + yPrev_s;

    xPrev_s = xd_s(i);
    yPrev_s = yd_s(i);
    thetaPrev_s = thd_s(i);
end

% Print out paths
hold on
plot(xd_m, yd_m, 'r');
plot(xd_s, yd_s, 'b');
scatter(xd_s(n-1), yd_s(n-1), 'b')
scatter(xd_m(n), yd_m(n), 'r')
% Create xlabel
xlabel({'X Position (meters)'});
% Create ylabel
ylabel({'Y Position (meters)'});
% set bounds and ratio
axis([0.6 1 -0 0.8])
axis('square')
```

```
% deviatedPursuit.m
% This script simulates a holonomic ground platform moving along a
% circular path with a slave unit following under the control of
% the deviated pursuit guidance laws.
% The a subset of the paths for both system is printed to the
% screen as a figure
```

```matlab
clear; clc;

% Independent variable - time
tstep = 0.01;
t = 0:tstep:pi;

% Desired x, y , th, u and w
xd_m = cos(t);
yd_m = sin(t);

dxd_m = diff(xd_m);
dyd_m = diff(yd_m);
thd_m = atan2(dyd_m, dxd_m);

ud_m = (dxd_m.^2 + dyd_m.^2).^0.5;
wd_m = diff(thd_m);

[m,n]=size(xd_m);

xPrev_s = 0.9;
yPrev_s = 0;
thetaPrev_s = 0;

for i = 1:1:(n-1),

    sigma_i_ip1 = atan2((yd_m(i) - yPrev_s) , (xd_m(i) - xPrev_s));

    r_i_ip1 = ((yd_m(i) - yPrev_s).^2 + (xd_m(i) - xPrev_s).^2).^0.5;

    alpha_ip1 = deg2rad(-1);

    if alpha_ip1 > deg2rad(1)
        alpha_ip1 = deg2rad(1);
    end

    if alpha_ip1 < deg2rad(-1)
        alpha_ip1 = deg2rad(-1);
    end

    vFollow = 1;

    dxd_s(i) = vFollow * cos(sigma_i_ip1 + alpha_ip1 );
    dyd_s(i) = vFollow * sin(sigma_i_ip1 + alpha_ip1 );

    wd_s(i) = ud_m(i) * sin(thd_m(i) - sigma_i_ip1) / r_i_ip1 ;

    thd_s(i) = wd_s(i) + thetaPrev_s;

    xd_s(i) =  dxd_s(i) * tstep + xPrev_s;
```

```matlab
    yd_s(i) =  dyd_s(i) * tstep + yPrev_s;

    xPrev_s = xd_s(i);
    yPrev_s = yd_s(i);
    thetaPrev_s = thd_s(i);
end

hold on
plot(xd_m, yd_m, 'r');

plot(xd_s, yd_s, 'b');

% set bounds and ratio
axis([0.95 1 0.1 0.3])
axis('square')

% Create xlabel
xlabel({'X Position (meters)'});

% Create ylabel
ylabel({'Y Position (meters)'});
```

```matlab
% proportionalPursuit.m
% This script simulates a holonomic ground platform moving along a
% circular path with a slave unit following under the control of
% the proportional pursuit guidance laws.
% The a subset of the paths for both system is printed to the
% screen as a figure.
clear; clc;

% Independent variable - time
tstep = 0.01;
t = 0:tstep:pi;
K = 0.99;

% Desired x, y , th, u and w
xd_m = cos(t);
yd_m = sin(t);

dxd_m = diff(xd_m);
dyd_m = diff(yd_m);
thd_m = atan2(dyd_m, dxd_m);

ud_m = (dxd_m.^2 + dyd_m.^2).^0.5;
wd_m = diff(thd_m);

[m,n]=size(xd_m);
```

```
xPrev_s = 0.9;
yPrev_s = 0;
thetaPrev_s = 0;

for i = 1:1:(n-1)

    sigma_i_ip1 = atan2((yd_m(i) - yPrev_s) , (xd_m(i) - xPrev_s));

    r_i_ip1 = ((yd_m(i) - yPrev_s).^2 + (xd_m(i) - xPrev_s).^2).^0.5;

    vFollow = 1;

    dxd_s(i) = vFollow * cos(K * sigma_i_ip1);
    dyd_s(i) = vFollow * sin(K * sigma_i_ip1);

    wd_s(i) = K * (ud_m(i) * sin(thd_m(i) - sigma_i_ip1) / r_i_ip1) ;

    thd_s(i) = wd_s(i) + thetaPrev_s;

    xd_s(i) =  dxd_s(i) * tstep + xPrev_s;
    yd_s(i) =  dyd_s(i) * tstep + yPrev_s;

    xPrev_s = xd_s(i);
    yPrev_s = yd_s(i);
    thetaPrev_s = thd_s(i);
end

hold on
plot(xd_m, yd_m, 'r');

hold on
plot(xd_s, yd_s, 'b');

% set bounds and ratio
axis([0.95 1 0.1 0.3])
axis('square')
% Create xlabel
xlabel({'X Position (meters)'});

% Create ylabel
ylabel({'Y Position (meters)'});
```
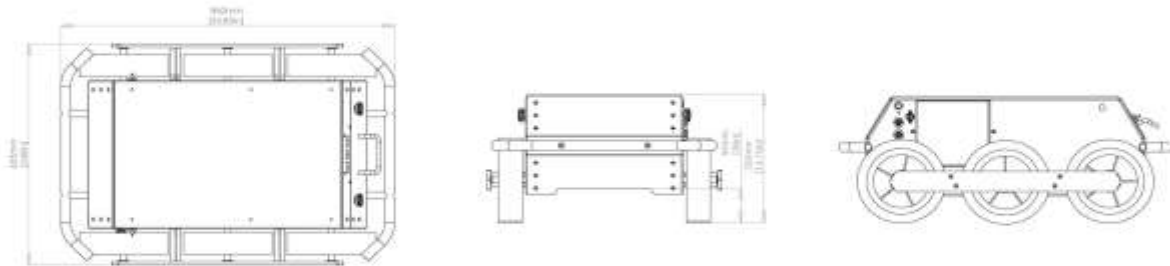
HUSKY A100™
UNMANNED GROUND VEHICLE

CLEARPATH
ROBOTICS™

HUSKY TECHNICAL SPECIFICATIONS

L x W x H	860x600x340 mm	POWER	600 W (2 motors)
CLEARANCE	85 mm (at centre)	MAX. SPEED	1.5 m/s
WEIGHT	35 kg	KINEMATICS	Differential drive
MAX. PAYLOAD	40 kg	OPERATING TIME	2 h
BATTERIES	12 V, 44 Ah		

specifications subject to change.

ASK US ABOUT OUR OTHER UNMANNED VEHICLE SOLUTIONS
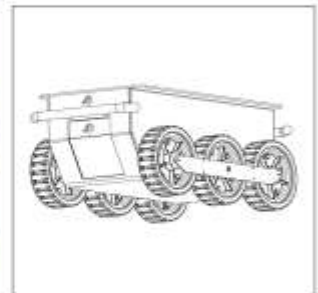

KINGFISHER M100   RHINO C100   CHAMELEON R100   CUSTOM VEHICLES

Find out if Clearpath Robotics can help make your next robotics project a success.
Call us to set up a free, 30 minute, no obligation consultation. We would love to hear from you.

Clearpath Robotics Inc.
Waterloo Research and Technology Park, 295 Hagey Blvd., Waterloo, Ontario, N2L 6R5, Canada
www.clearpathrobotics.com | info@clearpathrobotics.com | Toll Free: 1-800-301-3863 | Local: (519) 513-2418

8 WORKS CITED

[1] S. Magnuson, "Army, Car Makers Push Ahead With Driverless Vehicle Research," *National Defence Magazine,* August 2011.

[2] G. Richards, "Intelligent cars," *Engineering & Technology,* vol. 5, no. 1, pp. 40-14, 2010.

[3] J. Davis, A. Animashaun, E. Schoenherr and K. McDowell, "Evaluation of semi-autonomous convoy driving," *Journal of Field Robotics,* vol. 25, no. 11-12, p. 880–897, 2008.

[4] K. Bullis, "How Vehicle Automation Will Cut Fuel Consumption," *Technology Review,* 24 October 2011.

[5] G. B. &. K. A. Patrick Lin, "Autonomous Military Robotics: Risk, Ethics, and Design," Ethics + Emerging Sciences Group, California Polytechnic State University, San Luis Obispo, 2008.

[6] "Lockheed Martin Demonstrates New Ambush-Thwarting Push-Vehicle Capability for Automated Convoy Program," PR Newswire, Dallas, 2010.

[7] M. D. A. Green, "The Future of Autonomous Ground Logistics: Convoys in the Department of Defense," School of Advanced Military Studies, Fort Leavenworth, Kansas, 2011.

[8] G. Jean, "Army, Marine Corps In Pursuit of Robotic Convoy Systems," *National Defense Magazine,* December 2010.

[9] "First demonstration of SARTRE vehicle platooning," 17 January 2011. [Online]. Available: http://www.sartre-project.eu/en/press/Documents/Press%20release%2020110117%20First_test_platooning%20doc.pdf. [Accessed 2 March 2012].

[10] J. Ghommam, H. Mehrjerdi and M. Saad, "Leader-Follower Formation Control of Nonholonomic Robots with," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, 2011.

[11] P. Petrov, "Nonlinear Adaptive Control of a Two-Vehicle Convoy," *The Open Cybernetics and Systemics Journal,,* vol. III, pp. 70-78, 2009.

[12] P. Petrov and O. Boumbarov, "Nonlinear Adaptive Control of a Two-Vehicle Autonomous Convoy Using a Look-Ahead Approach," in *WSEAS International Conference on SIGNAL PROCESSING, ROBOTICS and AUTOMATION*, Cambridge, 2008.

[13] M. E. E. N. &. F. C. Cherif Smaili, "A Road Matching Method for Precise Vehicle," *Journal of

]   *Intelligent Transportation Systems,,* pp. 76-188, 2008.

[14   F. B. &. B. Belkhouche, "Modeling and Controlling a Robotic Convoy Using Guidance Laws
]   Strategies," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS,* pp. 813-825, 2005.

[15   F. Chaumette, P. Rives and B. Espiau, "Positioning of a robot with respect to an object, tracking
]   it and estimating its velocity by visual servoing.," in *Proc. IEEE Int. Conf. Robotics and
        Automation*, Palo Alto, 1991.

[16   S. Feyrer and A. Zell, "Detection, tracking, and pursuit of humans with an autonomous mobile
]   robot," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Kyonglu, Korea, 1999.

[17   S. Feyrer and A. Zell, "Tracking and pursuing persons with a mobile robot Faces and Gestures in
]   Real-time Systems,," in *Proc. Int. Workshop on Recognition, Analysis and Tracking*, Los Alamitos,,
        1999.

[18   M. D. H. M. A. Bazoula, "Formation Control of Multi-Robots via Fuzzy Logic Technique," in *Int. J.
]   of Computers, Communications & Control*, 2009.

[19   F. E. D. P. F. V. E. S. I. D. Carlos Santos, "Fuzzy Decentralized Control for Guidance of a Convoy of
]   Robots in Non-linear Trajectories," IEEE, 2010.

[20   K. C. Cheok, G.-E. Smid and K. Kobayashi, "A Fuzzy Logic Hierarchical Intelligent Control System
]   Paradigm for an In-Line-Of-Sight Leader-Following," *Journal of Robotic Systems,* vol. HMMWV,
        1999.

[21   E. B. &. A. Y. N. Morgan Quigley, "STAIR: Hardware and Software Architecture," Computer
]   Science Department, Stanford University, Palo Alto.

[22   P. Petrov, "A Mathematical Model for Control of an Autonomous Vehicle Convoy," *WSEAS
]   TRANSACTIONS on SYSTEMS and CONTROL,* vol. III, no. 9, pp. 835-848, 1991.