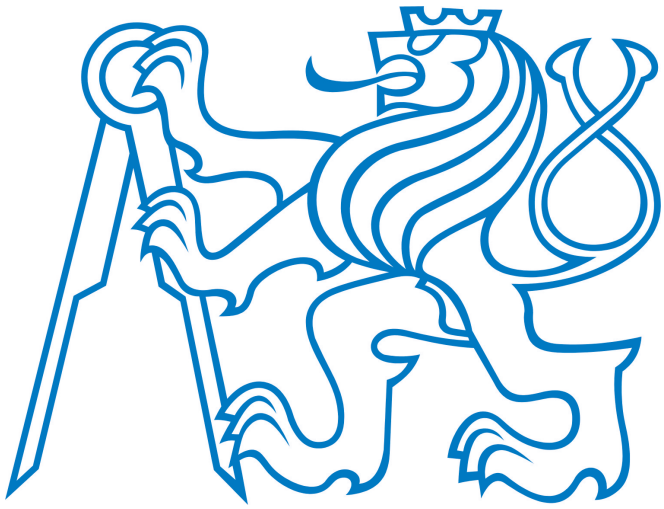
THE CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering

BACHELOR'S THESIS



Miodrag Ignjatovic

Platooning with Low-Cost Sensors

Project supervisor: Dr. Gaël Ecorchard

Department of Control Engineering
Major: Cybernetics and Robotics
Specialization: Systems and Control

January 2017

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praye dne …..... …..

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

BACHELOR PROJECT ASSIGNMENT

Student: **Miodrag Ignjatovic**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Bachelor Project: **Platooning with Low-Cost Sensors**

Guidelines:

An intermediate step between current technology and fully-autonomous vehicles are platooning solutions, where a human drives a leading vehicle while autonomous vehicles follow the same path.
The aim of this work is to combine well-tested technologies from the IMR Group to allow for platooning with camera(s) as sole sensor.
1. Development of a relative localization system with a camera and a pattern detector algorithm to determine the position of the follower vehicle relative to its guide.
2. Development of a path following algorithm based on the odometry of both guiding and guided vehicles
3. Participation in the development of a path following algorithm based on GPS data of both guiding and guided vehicles.
4. Participation in the development of a combining algorithm for all localization algorithms.
5. Writing of a final report.

Bibliography/Sources:

[1] Bergenheim, C.; Shladover, S. & Coelingh, E. Overview of Platooning Systems Proceedings of the 19th ITS World Congress, 2012 .
[2] Ali, A.; Garcia, G. & Martinet, P. Safe Platooning in the Event of Communication Loss using the Flatbed Tow Truck Model Proceedings of the 13th International Conference on Control, Automation, Robotics & Vision, 2014.
[3] T. Krajník, M. Nitsche et al.: External localization system for mobile robotics. International Conference on Advanced Robotics (ICAR), 2013.

Bachelor Project Supervisor: Pierre Gael Ecorchard Marie Dr.

Valid until the summer semester 2016/2017

L.S.

prof. Ing. Michael Šebek, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 11, 2016

Acknowledgements

I would like to express gratitude to my thesis supervisor Dr. Gaël Ecorchard for his guidance and patience. Also, I would like to thank everyone in the IMR group as well as Czech Institute for Informatics, Robotics and Cybernetics. Finally I would like to thank my family for their inexhaustible support.

Abstract

The goal of this project is to design and implement a platoon system between two vehicles based on relative localization system and odometry path follower. The system utilizes quasi operating system called Robot Operating System for the purpose of communication between vehicles and computation of relative parameters. Relative localization is based on a vision-based external localization system called Whycon. Furthermore, this thesis is about investigation of what can be achieved when combining low cost sensors in order to achieve better results. Extended Kalman Filter was used to combine wheel odometries of vehicles and Whycon localization in order to overcome their individual problems.

Keywords: Platoon system, Relative Localization, Odometry path follower, Robot Operating System, Odometry combination.

Abstrakt

Cílem toho projektu je návrh a implementace systému konvoje dvou vozidel založeném na relativní lokalizaci a sledování trajektorie z odometrii. Systém používá kvazi operační systém Robot Operating System pro komunikaci mezi vozidly a určení relativních parametrů. Určení relativní polohy vozidel je realizováno externím systémem Whycon, který zpracovává obrázky z kamery. Tato práce rovněž zkoumá možnosti použití levných základních senzorů pro zlepšení výsledků. Byl použit rozšířený Kalmánův filtr pro fúzi odometrických dat a lokalizace ze systému Whycon pro zlepšení výsledků, které produkují jednotlivé metody samostatně.

Klíčové slova: konvoj, relativní lokalizace, sledování trajektorie z odometrie, Robot Operating System, kombinace odometrie

1. Contents

2. Introduction

In recent years many different car manufacturing companies have started to develop and even produce their own brands of autonomous cars. This technology, even thou it seems already available and viable, has a lot more to go through until it becomes part of our daily interaction with technology. This is why many intermediate steps will have to be taken before we can achieve fully automated vehicles.

In order to properly define any intermediate step, certain premise has to be taken into account. This is done in order to give breathing space for other, perhaps smaller problems that will have to be solved in order to achieve better systems. The main premise to which this project has been oriented around is the fact that a human driver is irreplaceable. However, behavior of a human driver can be emulated by an autonomous vehicle. From these two assumptions a system can be formulated such that a human driver would be in control of a guiding vehicle, while an autonomous vehicle can be guided along the same path taken as the guiding vehicle. This solutions is usually referred to as a **Platooning Systems**.

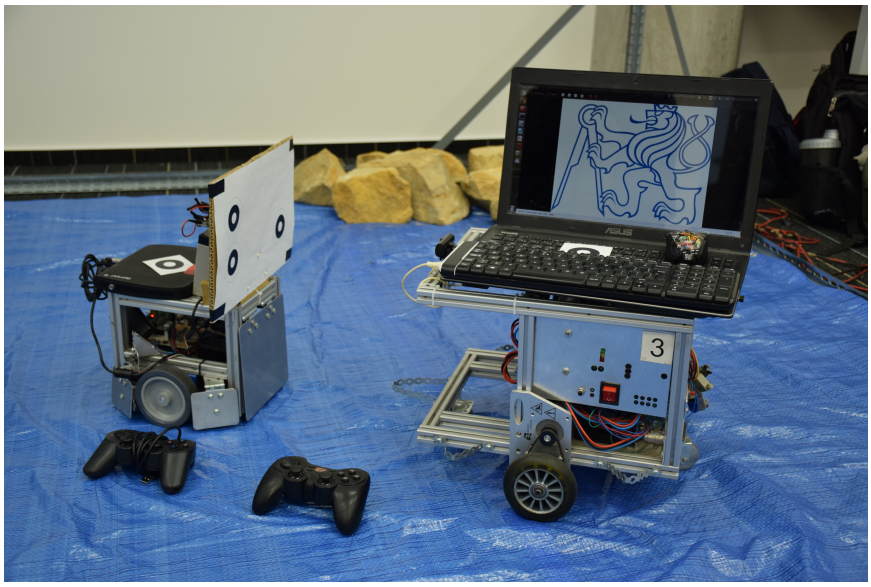


Figure 2.1.1. A Platoon of Robots made by the IMR group

A **Platooning System** can be defined as a collection of vehicles that travel together, while actively coordinated in formation. Algorithms and systems that are being designed to create platoons are not trivial at all. They are going to improve upon fuel consumption, safety, traffic efficiency, comfort as well as

potentially change the way we think about out transportation systems. Another significant factor which makes this project relevant to the future of autonomous vehicles is the fact that one way or the other, platooning systems will be incorporated into fully automated cars. This is because platooning revolves around cooperation between vehicles and with autonomous vehicles this cooperation will lead to better solutions and better utilization of our transportation systems.

Platooning has been developed by many researchers. Some of these are European platooning project (SARTRE), California traffic automation program that includes platooning (PATH), Cooperative driving initiative (GCDC) and Energy ITS (Japanese truck platooning project) [1] [2].

In order for any platooning algorithm to work, a good localization system must me developed in order to determine relative positions of the following and leading vehicle. There is wide range of these algorithms and even wider range of the types of sensors they use, but for the purposes of my thesis I have decided to use a visual-based external localization algorithm called **Whycon**.

Another important part of this thesis is the development of a **path following algorithm**. There are many different solutions of how the following vehicle could conduct its purpose, but the solution that I have implemented revolves around a path following algorithm based on the combined odometry information from wheels, localization algorithm and GPS

It is important to note that most all of my work will be conducted in computer simulated environments, but it will also include implementation of a platooning system on real robots robots provided by the IMR group of Czech Institute for Informatics Robotics and Cybernetics (Figure 2.1.1.). The main software tool used in this project is an *open source* computer system infrastructure called **Robot Operating System**.

I firmly believe in the power of open source concept. And since all of the work don't on this thesis has been on open source based computer programs I would like to share some of my thoughts about them. Open source programs and projects give future engineers like myself the necessary tools and freedom with which we can built upon existing technologies. The community of open source users provides a great support and knowledge in our endeavors. My beliefs and ideology around open source concept goes beyond the scope of this project, but it is important to know that because of open source projects developed by people around the world I was empowered to do my bachelor thesis.

3. Robot Operating System (ROS)

Robot Operating System is a quasi operating system designed to provide the necessary software infrastructure for software developers who focus on different aspects of robotics. Since one of the primary goals of ROS is standardization and reuse of programs, a good and large community of users was organized in order to share, instruct and help fellow engineers.

This network has provided me with helpful instruction as well as some code ideas and packages which I have implemented in my bachelor work. The main aspect of what makes ROS so versatile and useful is its utilization of a distributed system composed of ROS nodes, ROS topics and ROS messages. It is important to understand them, and in this project they will be referred to as just nodes, topics and messages.

ROS messages

ROS messages [8] are standardized and simplified forms of data values which are to be transported across systems.

ROS topics

ROS topics [9] are named buses over which ROS nodes can exchange messages. In general, ROS nodes are not aware of who they are communicating with. Instead, nodes that are interested in certain data subscribe to the relevant topic and nodes that generate data publish to the relevant topic.

ROS nodes

ROS nodes [10] are processes that preform computation. Code complexity is reduced and systems can be easily mapped so that they make more sense. ROS nodes are written with ROS client library, such as **roscpp** and **rospy**. For the purposes of this project, I have used roscpp, as it is a C++ implementation in ROS.

Gazebo

Gazebo [11] is robot simulation tool. It is equipped with a physics engine and graphical interface. It is integrated into ROS. For the purposes of this project, a 2D surface plane was simulated in Gazebo. All of my simulations have been through gazebo.

4. Robot Model

The main idea behind creation of the model I used in the simulation and testing of the algorithms I was required to build, was to develop as simple of a model as possible to which I can continuously add more features that I felt were required. Since I also wanted to test algorithms on real machines (that were provided by the IMR group), robot model had to faithfully represent characteristics of these machines as well. Therefore robot model produced was created in the struggle to satisfy previously mentioned ideas.

I did not want to get too involved in creating visually faithful representations of provided robots, or to get too pedantic on physical parameters (such as size, mass, inertias ...) of the provided machines. This is why my model and the machines provided do not look alike. The more important aspect that I put my focus on was for the model and provided machines to behave in the same manner when put in the same conditions. In order for this to be a realistic desire, throughout this project, I was trying to identify crucial parameters that could be easily adjustable in order to match behavior of the model to the behavior of robots.

It has to be said that all this was made a lot easier because of the way Robot Operating System (ROS) works with Gazebo and Unified Robot Description Format (URDF) models.

4.1. Unified Robot Description Format (URDF)

Unified Robot Description Format (URDF) [12], is an XML file format used in ROS in order to describe different elements of a robot. URDF files can also be simulated in Gazebo (some additional simulation-specific tags must be specified in order to work properly). While URDF files are a useful and standardized format in ROS, they are lacking many features and have not been updated to deal with the evolving needs of robotics. However for purposes of this project URDF models worked perfectly. It is important to mention that URDF models can only specify the kinematic and dynamic properties of a single robot in isolation. However ROS provides the ability to replicate the same model multiple times. URDF can not specify the pose of the robot itself within a world, this is the job of Gazebo and ROS. It is apparent that URDF models, Gazebo and ROS do not provide much when used separately from each other, but when combined together they make a very powerful simulation tool, and all of the previously

outlined problems of URDF models have been rendered irrelevant.

Another tool that was used in order to add onto URDF model was Xacro (XML Macros). Xacro [13] is an XML micro language, and for the purpose of this project it was used in order to simplify the robot description. Specifically it was very helpful when some design changes were required.

The way URDF models are created is by defining different **links** of the model and how they are **joint** together. Primary characteristics of links are visual and collision geometries of parts and their inertias, while primary characteristics of joints are the type of joint in question, parent link, child link and origin (position) where links are joint. In this sense, links of URDF models are inherently hierarchical.

Basic links of any model are **Base Link**, **Base Footprint** and **Internal Link** of the robot modeled. Base link of a model functions as a *chassis* of the robot. Base link dose not joint to any other link because other links are joint to base link. Base link is also the main link through which Gazebo and ROS utilize URDF models. Base footprint of a model is the point that is on the surface directly underneath the center of the base link. Internal Link of the model stores its internal properties (like inertia). Both base footprint and internal link are joint to the base link with a *fixed* joint type. Robot model files are defined in **Platoon Description** package.

4.2. Drive System

This was perhaps the most crucial part of the model design as it has massive influence on so many other aspects of this project, like steering system. However, this is also one of the aspects in which I had to assimilate provided robots with my model. Provided machines were of **differentially wheeled robot** type, and so are my models.

Differentially wheels robots that were provided had left and right side wheel, as well as caster wheels for balance. For the model purposes, left and right wheel link were created and joint onto base link. As the wheels are suppose to turn, the joint type here is continuous. Continuous joint types, require *transmission* to be specified. This was defined as simple transmission. Since wheels are not passive, but active parts of the model, Gazebo reference had to be specified for both wheels. Also in order to control wheels, **Gazebo plugin** [14] called **Differential Drive Controller** was set up. Final step is to activate

the controller with **controller manager** [15] ROS package and to start publishing states using **robot state publisher** [16] package. These two packages along with yaml file (specifying further attributes of the controller) form **Platoon Control** package.

Platoon control package opens **cmd_vel** topic through which commands are sent to specify linear and angular velocities that the robot is suppose to make. These are known as **twist** commands. This means that after specifying all necessary parameters of differential drive controller I did not have to focus on speeds and accelerations of specific wheels, but to focus on velocities of the model as a whole. Platoon control package also initiates odometry topic which is essential for this project. However, real robots do not need platoon control package, as specific packages were also provided for them, witch preform these functions. These are **Morbot** and **ER1** packages. Because of this transition from simulation to realization was simple as far as this aspect was concerned.

In order to control the leading robot I have decided to use a simple joystick. **Teleop twist joy** [17] package provided me with the necessary transition between joystick commands and twist commands. I also created additional package in order to specifically define different functions I wanted to get out of different buttons on the joystick [18]. This package also offer much better communication with ROS than packages implementing control from a keyboard. This package is called **Platoon Teleop** package.

4.3. Camera and Patterns

Model of a camera was made using another gazebo plugin called **camera controller**. Unlike some parts of the model, I felt like camera had to be accurately modeled on the real camera used in the real machine. Since this project is about platooning with low-cost sensors, camera selected was a simple **UVC camera**. A ROS package was already written for a UVC camera [19] and was setup according to tutorial [20]. Camera also had to be calibrated since additional nodes were required for Whycon algorithm to function properly.

Another link was added to the robot model defining the shape and graphics of the pattern used by Whycon algorithm. This link, since it has to contain very specific graphics of patterns, it was designed by using open source programs: **Blender** [21] and **MeshLab** [22].

5. Kinematic Equations and Control Law

From the model described in the previous chapter, it is clear that since the vehicle is described by two independently controlled wheels such that linear and angular velocity of the vehicle can be controlled independently.

First step I took is to develop polar coordinate system to describe the kinematic equations. This is done because it is a much more intuitive way of thinking about the control problem, as it emulates human driving behavior.

To formulate the basic problem of the system, I supposed that there is a desired position and orientation different form the current position and orientation of the follower vehicle. The following three parameters become apparent and they describe the system absolutely in the polar coordinate system.

r	Distance between current and desired position
$\delta \in (-\pi, \pi]$	Smallest angle between current orientation and desired position
$\theta \in (-\pi, \pi]$	Smallest angle between current and desired orientation

This is called egocentric polar coordinate system with respect to observer. These can be graphically represented in Figure 5.1.1..

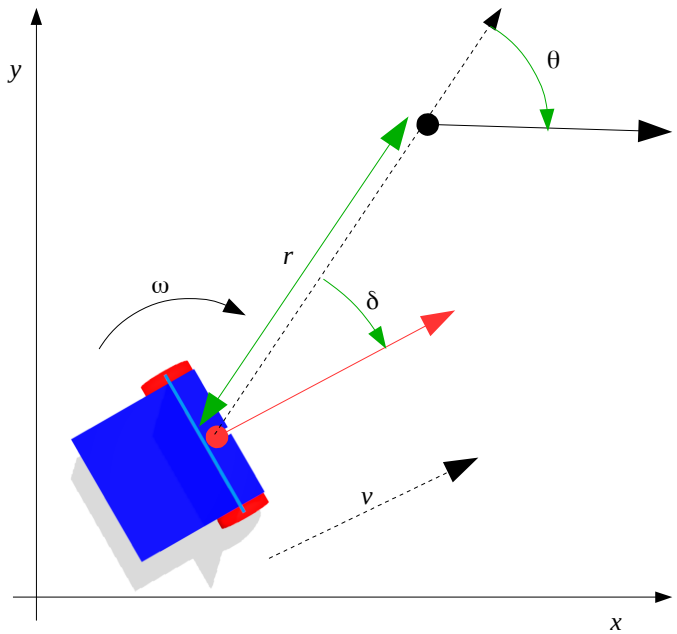


Figure 5.1.1. Egocentric Polar Coordinate System

Relationship between egocentric polar velocities and vehicle velocities can therefore be written as follows:

$$\begin{pmatrix} \dot{r} \\ \dot{\theta} \\ \dot{\delta} \end{pmatrix} = \begin{pmatrix} -v\ \cos(\delta) \\ \frac{v}{r}\ \sin(\delta) \\ \frac{v}{r}\ \sin(\delta) + \omega \end{pmatrix} \quad (1)$$

Where v and ω are linear and angular velocity of the vehicle.

5.2. Control Law for Angular Velocities

In the previous subsection the coordinates of error space was defined. In this part the control problem of moving a vehicle from given position and orientation to desired position and orientation becomes a problem of bringing the coordinates of error space to origin, or in other words: bringing r, δ and θ to zero.

The solution to this problem is described in [4] . In [4] this problem was solved by decomposing the system *(1)* into two subsystems:

$$\begin{pmatrix} \dot{r} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} -v\ \cos(\delta) \\ \frac{v}{r}\ \sin(\delta) \end{pmatrix} \quad (2)$$

$$\dot{\delta} = \frac{v}{r}\ \sin(\delta) + \omega \quad (3)$$

The idea is to find a virtual control, δ which steers the subsystem *(2)* (vehicle position) to the origin and the real control, ω which renders the dynamics of the subsystem *(3)* sufficiently faster than subsystem *(2)* and stabilizes δ quickly to the desired virtual control.

Finally, the control law proposed by [4] for ω is:

$$\omega = -\frac{v}{r}\left[k_2\ (\delta - \arctan(-k_1\theta)) + \left(1 + \frac{k_1}{1 + (k_2\theta)^2}\right)\ \sin(\delta)\right] \quad (4)$$

Where k_1 and k_2 are positive constants. It is worth to say that this control law dose not put any restrictions on v, other than to be a nonzero

positive. This also means that ω is now a linear function of v, since:

$$\omega = k(r,\theta,\delta)\ v$$

Where k is the curvature of the path resulting from proposed control law, modified by k_1 and k_2, where k_1 is the ratio of the rate of change in θ to the rate of change in r (shown in Figure 5.2.1.) and k_2 modifies the general shape of the path as shown in Figure 5.2.2.

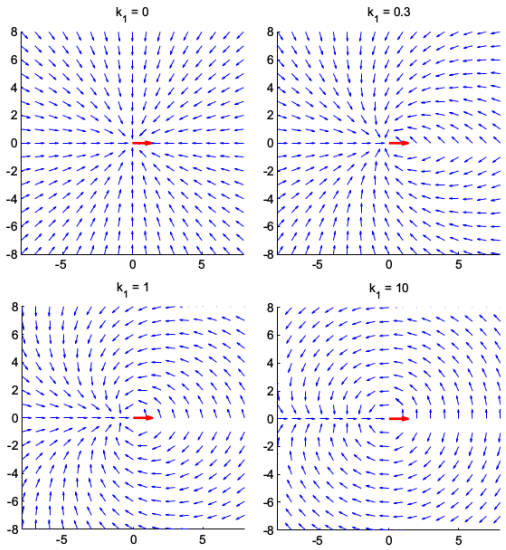


Figure 5.2.1. Example of how k_1 modifies the path taken

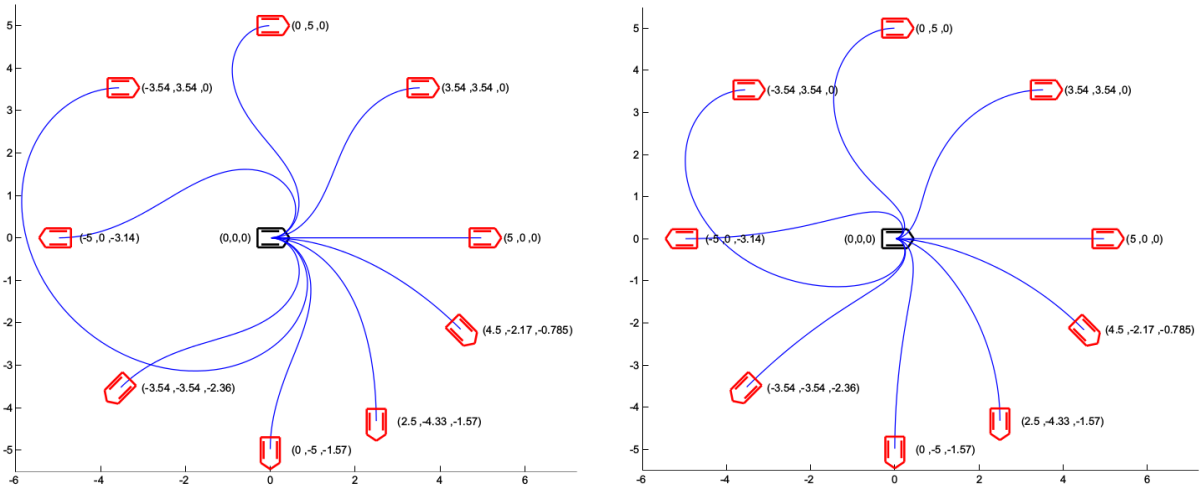


Figure 5.2.2. Example of how k_2 modifies the path taken

The curvature of a path of a vehicle moving on a plane is simply ω/v . So we can write:

$$k = -\frac{1}{r}\left[k_2\ (\delta - \arctan(-k_1\theta)) + \left(1 + \frac{k_1}{1 + (k_2\theta)^2}\right)\sin(\delta)\right] \quad (5)$$

Geometrically speaking, the path given by the virtual control is the Archimedean spiral as shown in the previous diagram. This also implies that the shape of the path taken by the vehicle is not influenced by the choice of v. This is true for the problem for which this control algorithm is designed but not true for my problem. The main difference being the fact that in my problem formulation, the target could dynamically change all the time, while in [4] the target is static. This became apparent while testing these algorithms. The main idea behind the following change was that the angular velocity should not be influenced by the distance r. This was made possible by switching the parameter v with the distance r in the equation for the angular velocity *(4)*. This change results in removing parameter r from control law for angular velocity since the new function is:

$$\omega = -\left[k_2\ (\delta - \arctan(-k_1\theta)) + \left(1 + \frac{k_1}{1 + (k_2\theta)^2}\right)\sin(\delta)\right] \quad (6)$$

This solution was tested and it worked much better than one proposed by [4] for my problem. Instinctively I was worried that this change might affect the shape of the path taken by the vehicle, however this is not a problem since parameter k_1 is the ratio of the rate of change in θ to the rate of change in r.

5.3. Control Law for Linear Velocity

The proposed algorithm for the linear velocity controller is based on a PID controller where the dynamically minimized error signal is the error between the current and desired distances of the follower vehicle (r).

$$v(t) = K_p\ e(t) + K_i\int_0^t e(\tau)d\tau + K_d\frac{de(t)}{dt} \quad (7) \qquad \text{Where:} \quad e(t) = L - r(t)$$

Where parameter L is the longitudinal referential distance that the follower robot must keep away from the leading robot.

PID controller

The PID control tools necessary for control program were provided by ROS **control toolbox** [23] package and the **PID controller** [24] was tuned manually using **Ziegler-Nichols' closed loop method** [6].

CO filter

The PID controller, especially the derivative part of the controller, produced noise and erratic control behavior. This is caused by the amplification of the measured process variable which is reflected as "chatter" in the controller output signal [25]. In order to create a better controller, a signal filter was added to the controller output as shown in Figure 5.3.1.

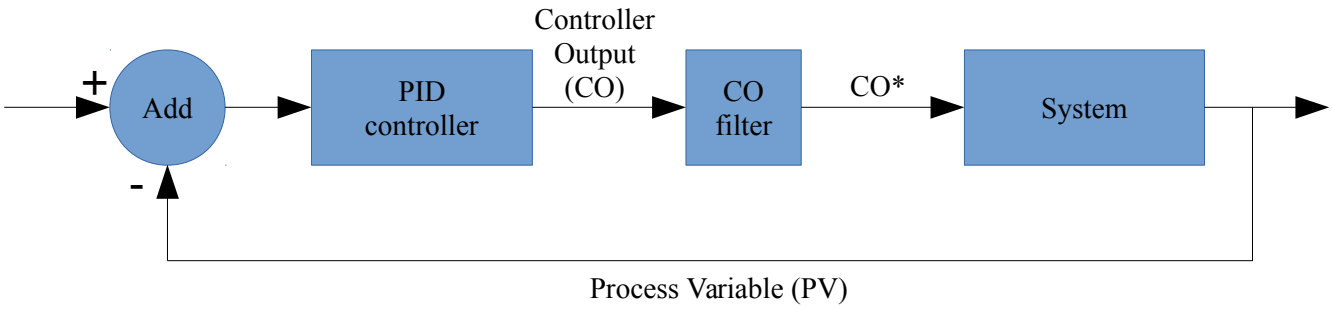


Figure 5.3.1. PID controller with Controller Output filter

For my controller output filter, I have chosen a simple first order filtering algorithm.

Continuous time filter:

$$T_f \frac{dCO^*}{dt} + CO^* = CO \quad (8)$$

Discrete time filter:

$$T_f \frac{(CO^*_{new} - CO^*_{old})}{(t_{new} - t_{old})} + CO^*_{old} = CO \quad (9)$$

or: $$CO^*_{new} = CO^*_{old} + \frac{(t_{new} - t_{old})}{T_f} (CO - CO^*_{old}) \quad (10)$$

Where CO is the raw controller output, CO* is the filtered controller output and T_f is the filter time constant (tuning parameter). In Figure 5.3.2. and

Figure 5.3.3. it can be seen how the first order filtering algorithm influences results of the linear velocity control algorithm.

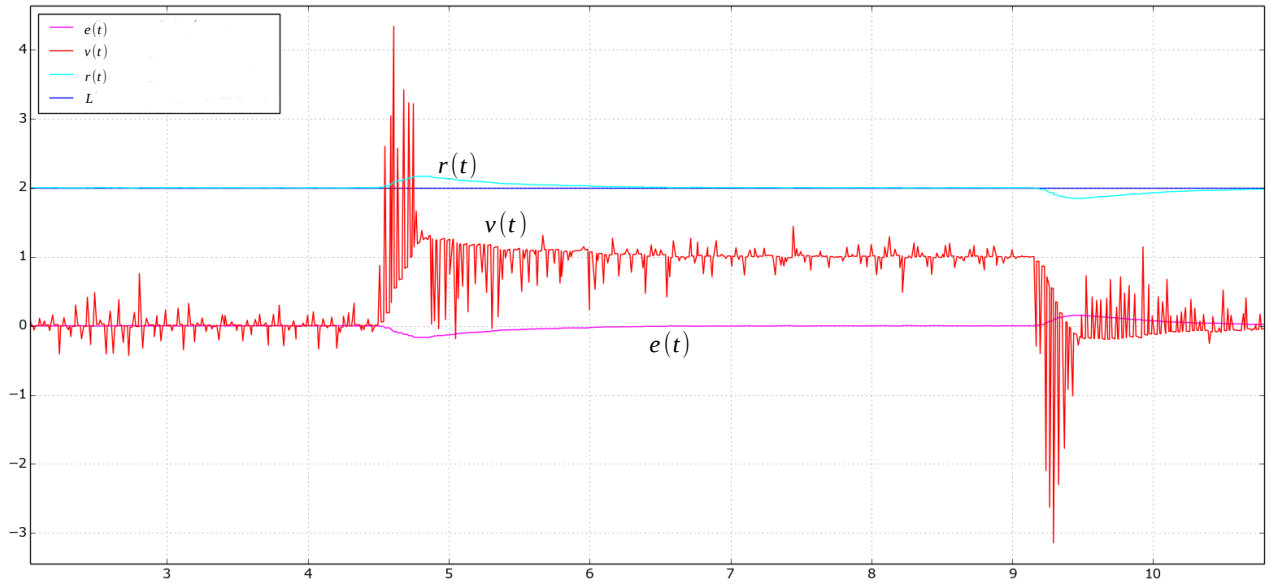


Figure 5.3.2. PID controller without CO filter

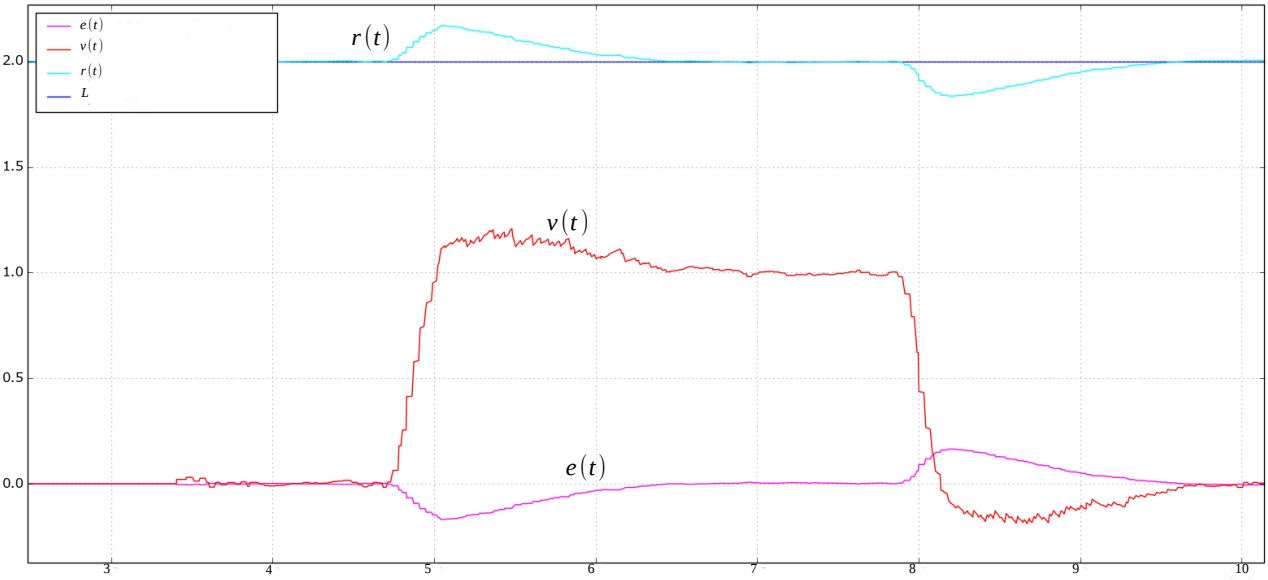


Figure 5.3.3. PID controller with CO filter

All controller algorithms were specified in **Platoon Algorithms** package under **odometry_follower_test_with_navigation** node.

6. Relative Localization System

The pattern detector algorithm that was used in this project is ***Whycon*** recognition system [3]. Whycon is a vision-based external localization system. It is a very precise, robust, efficient and low-cost system because it is able to utilize even the smallest and cheapest cameras to produce first class sensors.

6.1. How does Whycon work?

Whycon operates by processing images provided by a simple camera system and is based on a fast and precise detection of a black and white pattern, which consists of two concentric annuli with a white central disc. An initial detection step is performed to identify the position of the pattern in pixel coordinates. Using camera re-projection techniques and known dimensions of the inner and outer roundels, the three-dimensional position of the target with respect to the camera is computed. Finally, a transformation of the coordinate frame is applied to compute target coordinates with respect to a user-defined frame, either in three or two dimensions, depending on the chosen scenario. More detail can be found in [3]. Basic parameter of the Whycon algorithm can be found in Figure 6.1.1.

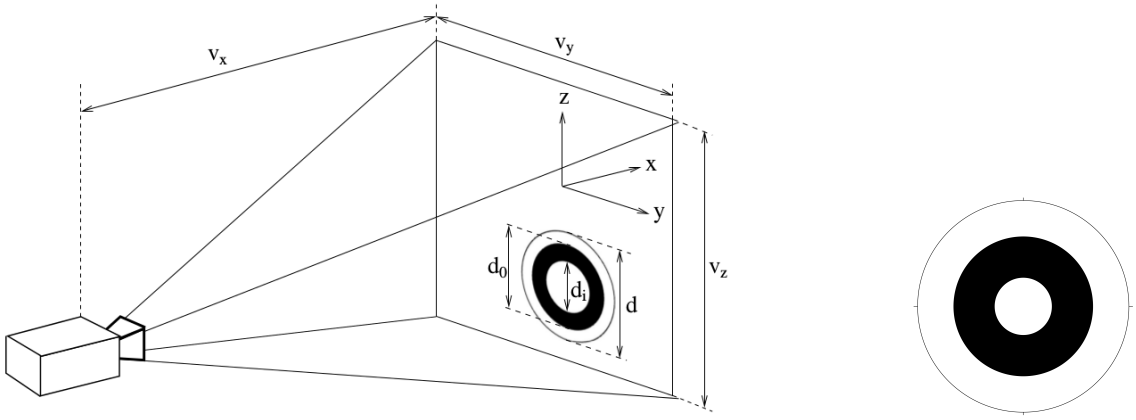


Figure 6.1.1. Whycon pattern localization and Whycon pattern/roundel shape

Whycon topics and messages

In practice, the whycon algorithm requires raw images to be processed and calibrated. This service is provided by the ROS package **image_proc** [26]. All connections can be simply understood from the RQT graph shown in Figure 6.1.2.

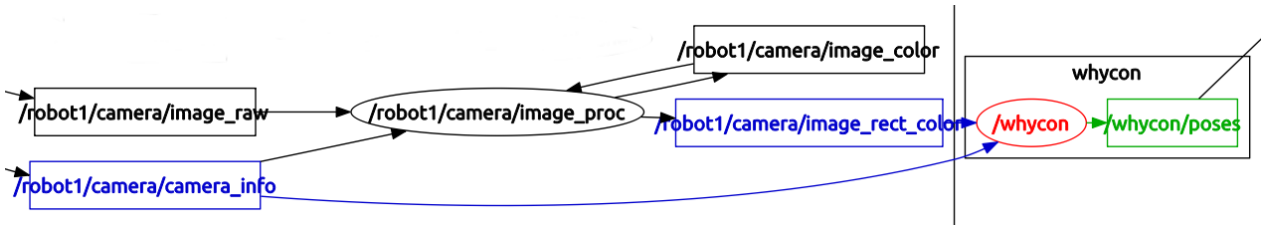


Figure 6.1.2. RQT graph for whycon and image_proc

The published topic of most relevance is the **poses** topic. Poses topic is composed of relevant positions of roundels. Each roundel is assigned an ID reference and each roundel position is defined by its position relative to the camera (x, y and z) as well as its orientation to the camera in the form of a quaternion (x, y, z and w). It is important to note that all roundels have to be the same size in order for whycon algorithm to recognize their position properly.

6.2. Limitations of Whycon algorithm

The Whycon algorithm outputs position (3D) and orientation (3D) of the pattern relative to the camera. However, it is not able to give any information on the “roll” around the pattern, as the pattern itself dose not change shape if it rotates around axes perpendicular to its plane. But whycon does provide the ability to recognize and output information about several patterns at once.

Because of this, I have decided to use three patterns in order to utilize the abilities of the algorithm. Using three patterns has also allowed me to ignore the output information of the orientation of the patterns, since I could easily and with greater precision use the point position information to calculate the orientations needed. The positions of roundels are show in Figure 6.2.1.

Figure 6.2.1. Arrangement of Roundels used

Some significant limitations of the whycon algorithm were discovered very quickly. The first main problem with whycon is that once any part of a roundel is

outside of the camera frame, the algorithm keeps sending the last recognized position of the roundel to the poses topic. As soon as a roundel is fully back in the frame it will resume working correctly. However, since I am using three roundels, another problem was encountered.

If roundels go out of the frame, and then come back into the frame, their ID referenced in the poses topic could be different than before exiting the frame. This is why it was very important to have the three roundels spaced out so that their distances in 3D space from each other would all be different. In this way roundels would always be defined by their relevant distances from each other.

Another problem that was encountered with whycon algorithm is the fact that as the yaw orientation of the robot is increased, precision of whycon algorithm worsens. Uncertainty of positions of patterns is increased. Luckily, this was easily solved by taking the average position of the roundels along all three axes. This step was also necessary as these averages represent the position of the leading robot relative to the camera. Figure 6.2.2. shows whycon limits regarding orientation of roundels.

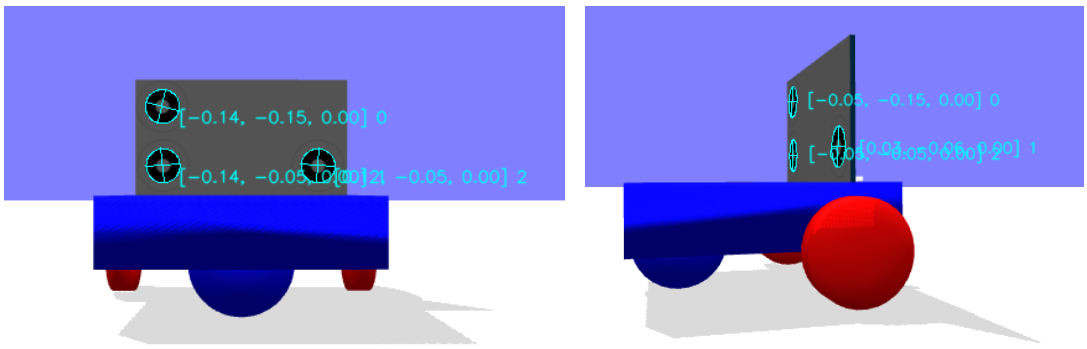


Figure 6.2.2. Whycon algorithm publishing "image out" topic at zero and at limit orientations

Finally there are basic limitations of the whycon algorithm based on how far away from camera can the algorithm recognize roundels. This limitation is influenced by the size of roundels as well as the quality of camera used. When combined with with limitations caused by the relative position of the pattern along the x axis (according to whycon) we get the area in which whycon can function relative to the follower robot (represented in Figure 6.2.3.)

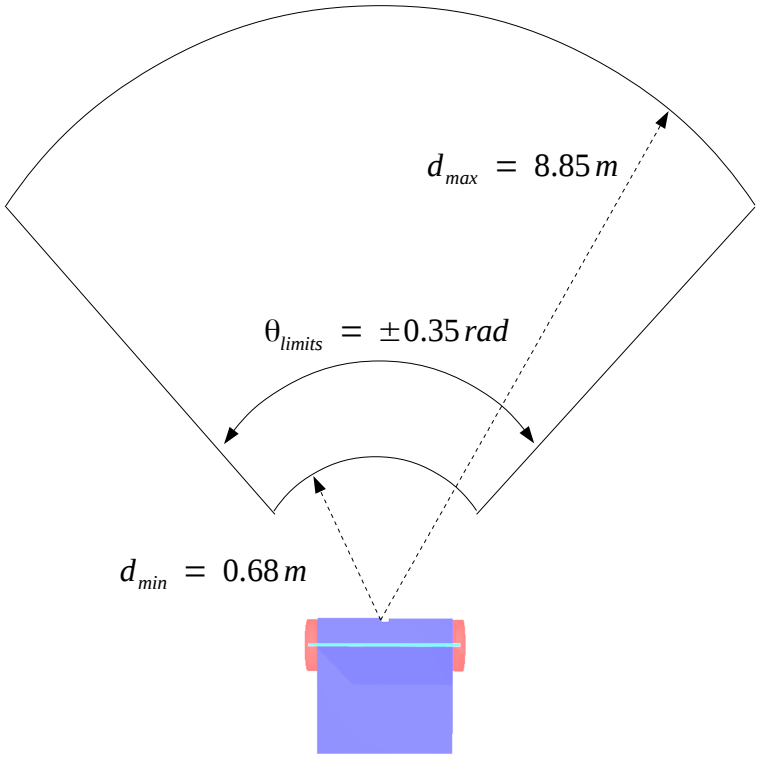


Figure 6.2.3. Geometrical Limitation of Whycon algorithm as used in simulation

This information is then used to calculate the optimal position where the leading robot should be located in order to provide the most flexible system.

6.3. Calculating Relative Positions of a Vehicles using Whycon

The callback function to whycon poses topic in the system provides us with a lot of useful information. First, in order to avoid confusion on whether whycon node is sending new information on the poses of roundels, time stamp given in the header of whycon poses topic is compared to the time stamp of the system. This provides a simple boolean parameter that will be useful in further implementation.

The second thing done in the callback function is the calculation and comparison of distances between poses of roundels in order to accurately know which pose corresponds to which roundel.

Finally, the relative position and relative orientation of the leader robot is calculated:

$$x_{rel} = (r_{x1} + r_{x2} + 2r_{x3})/4 \quad (11)$$

$$z_{rel} = (r_{z1} + r_{z2} + 2r_{z3})/4 \quad (12)$$

$$yaw_{rel} = \arctan\left(\frac{2r_{z3} - r_{z1} - r_{z2}}{2r_{x3} - r_{x1} - r_{x2}}\right) \quad (13)$$

Where x_{rel} and z_{rel} represent the position of the leader robot along x and z axis as given by whycon pose topic and r_{x1} , r_{x2} , r_{x3} , r_{z1} , r_{z2} and r_{z3} represent the positions of three roundels along x and z axis as given by whycon pose topic. Their ID (1, 2 and 3) is the same as pattern ID in figure 6.2.1. Finally, yaw_{rel} represents the orientation of the leader vehicle relative to the follower.

This piece of information is then used to calculate information needed in order to create odometry topic. Two different algorithms were developed to do this. The first one calculates odometry of the leader robot relative to follower robot (follower is set at origin on x-y plane with zero yaw angle), and second one calculates odometry of the follower robot to the leader robot (leader is set at origin on x-y plane with zero yaw angle). In both cases, delta and theta angles (as described in chapter on control algorithm for angular velocity) were calculated first. This information is then used to calculate the position of the robot on x and y axis as well as the yaw orientation on the x-y plane relative to relevant robot. Figure 6.3.1. shows the geometry of these calculations:

For leader relative to follower :

$$\delta = -\text{atan2}\left(\frac{-x_{rel}}{z_{rel}}\right) \quad (14)$$

$$\theta = \delta - yaw_{rel} \quad (15)$$

$$x = \sqrt{(z_{rel})^2 + (x_{rel})^2}\cos(\delta) \quad (16)$$

$$y = \sqrt{(z_{rel})^2 + (x_{rel})^2}\sin(\delta) \quad (17)$$

$$yaw = \delta + \theta \quad (18)$$

For follower relative to leader :

$$\delta = \text{atan2}\left(\frac{-x_{rel}}{z_{rel}}\right) \quad (19)$$

$$\theta = \delta - yaw_{rel} \quad (20)$$

$$x = -\sqrt{(z_{rel})^2 + (x_{rel})^2}\cos(\theta) \quad (21)$$

$$y = -\sqrt{(z_{rel})^2 + (x_{rel})^2}\ \sin(\theta) \qquad (22)$$

$$yaw = -(\delta + \theta) \qquad (23)$$

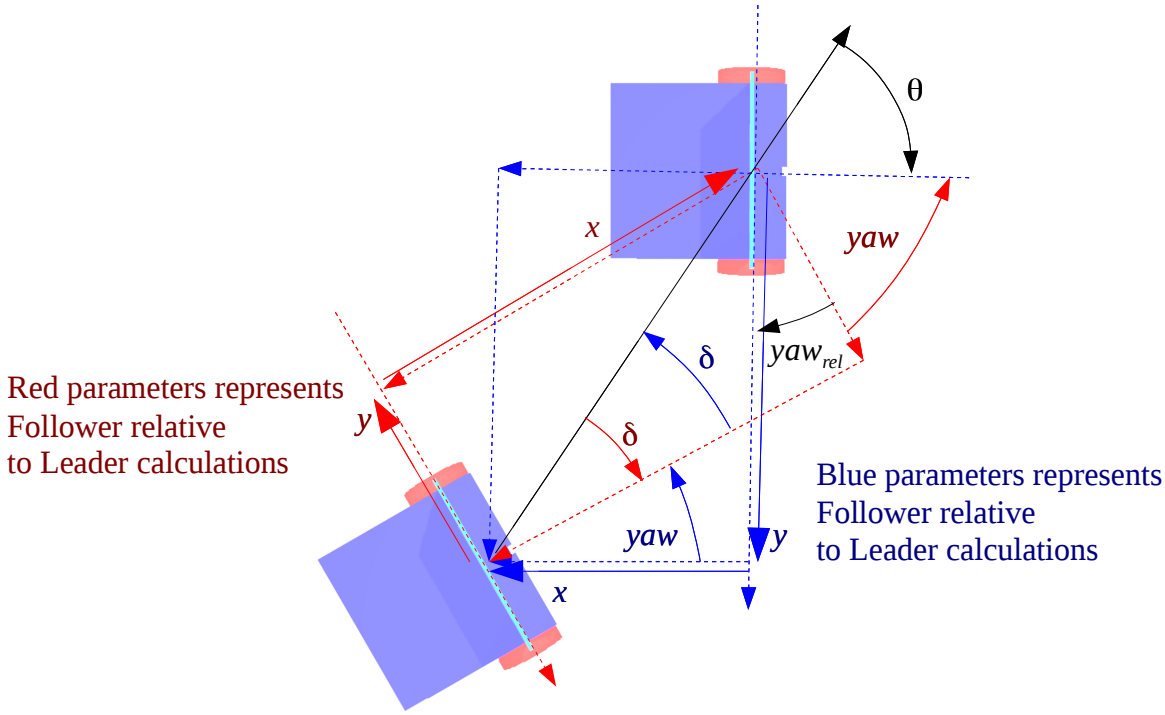


Figure 6.3.1. Relative parameters and coordinates of vehicles

Implementation of the solution to this problem was done in **Platoon Algorithms** package under **navigation** node.

7. Odometry Follower Algorithms

7.1. Position following algorithm

A penultimate step before designing my path following algorithm was designing a position following algorithm. In this project the position that is followed is given by the position and orientation of the leading vehicle as well as some fixed parameters.

In ROS, odometry navigation messages provide a subscriber with information on the position of a robot in 3D space (x, y and z coordinates), orientation of the robot (in the form of quaternion x, y, z and w) and twist velocities of the robot (x, y and z translational velocities, and x, y and z rotational velocities). Since this project is constrained to 2D space, robot positions of interest are going to be x and y coordinates of leading and following robot. Robot orientations of interest are yaw orientations of leading and following robot (on x-y plane).

For simulation purposes, ROS together with Gazebo provide odometry topics for spawned robots. For realization purposes, special packages were developed by the IMR group for robots they have made on which I have applied my algorithms. These packages are **morbot** and **er1**. All these packages provide odometry based on wheel movement.

The odometry callback function provides three important parameters that perfectly describe position and orientation of the robot on a x-y plane. These are:

x_L x_F position of leading and following robot on the x axis (world frame)

y_L y_F position of leading and following robot on the y axis (world frame)

yaw_L yaw_F angle between orientation of leading (and following) robot and x axis ($yaw \in (-\pi, \pi]$)

To create a successful position follower algorithm and to utilize already specified control algorithms, odometry topics from leader and follower robots are used to calculate r, δ and θ topics. An additional parameter was added in order to simplify the calculation. This is the angle which the line connecting centers of two robots makes with the x axis on the plane. This angle is referenced as α.

$$\alpha = \operatorname{atan2}\left(\frac{y_L - y_F}{x_L - x_F}\right) \qquad (24) \qquad \alpha \in (-\pi, \pi]$$

The following are the equations for calculating parameters needed for control algorithms:

$$r = \sqrt{((x_L - x_F)^2 + (y_L - y_F)^2)} \qquad (25)$$

$$\delta = \alpha - yaw_F \qquad (26)$$

$$\theta = \alpha - yaw_L \qquad (27)$$

Longitudinal reference distance that the follower robot must keep away from the leading robot (L mentioned in Kinematic Equations and Control Law chapter) is chosen by the user. However, this parameter will be closely tied in with the optimal working distance discussed in the whycon limitation chapter above. This is also the main parameter that is used to create the transformation from position of leading robot and desired position of the follower robot. It also becomes clear that the orientation of the follower robot is directly calculated from the orientation of the leading robot without any transformation. All this is visualized in Figure 7.1.1.

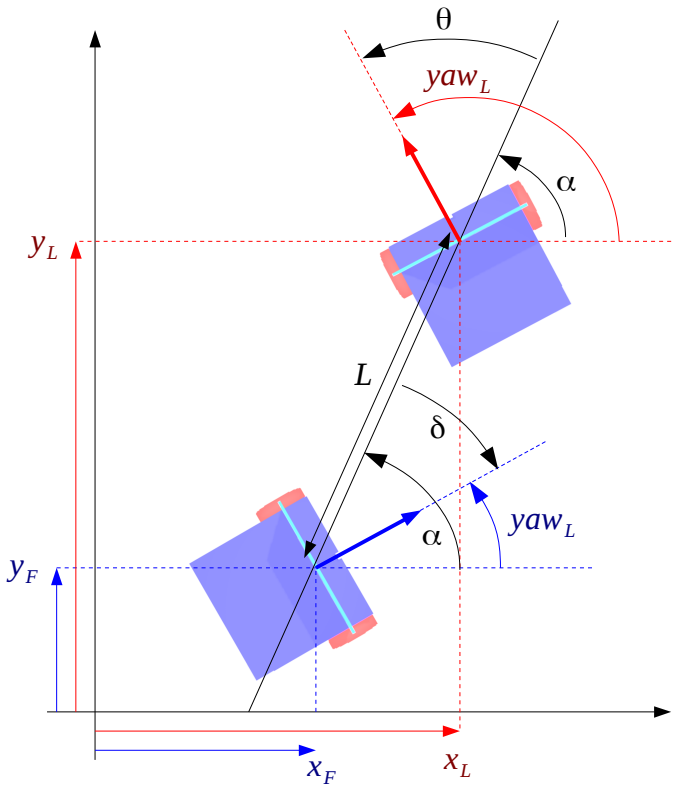


Figure 7.1.1. Position Follower Parameters

7.2. Path Following Algorithm

The main merit of the position following algorithm and its controllers is that they do follow positions continuously relatively accurately. The main fault was that there was no proper dynamic between follower robot and leader robot and as a result the path taken by the follower robot was much different that the path taken by the leader robot. To fix this fault and to keep the merits of previous algorithms, same odometry algorithm is used, but a point management algorithm was developed in order to make the follower robot keep to the path made by the leader robot. It is important to mention that odometry of both robots at the start is zero, but their initial positions and orientations are set by the user.

The path taken by the leader robot is basically a curve and all curves are collections of points. In order to make the follower robot keep to the path while using the position following algorithm the following path management algorithm was made:

1. Initialization of a list of points in 2D space based on x and y coordinates of the leader robot. The beginning point and only point on this list is the starting position of the leader robot based on the odometry of the leader robot.
2. The leader robot moves forward and if the distance between the end point on this list and the current position of the leader robot exceeds a given limit, **t1**, the current position of the leader robot is added to the end of the list and becomes the end point on the list.
3. If the list is empty, the current position of the leader robot is added to the list.
4. The x and y coordinates of the beginning point on the list are published
5. The follower robot moves forward and follows the beginning point on the list. If the distance between current potion of the follower robot and beginning point on the list is smaller than a given limit, **t2**, the beginning position of the list is removed.
6. if the list is empty, the current position of the leader robot is added to the list.
7. Algorithm returns to the second step.

In order for both position follower algorithm and path management algorithm to work properly more adjustments need to be made. The length of the path taken by the leader robot also has to be maintained since the distance between current positions of the leader robot and follower robot is not the real distance that the follower robot has to take. A good approximation of the path length is

calculated by incrementing the path length by the distance between current position of the leading robot and the end point on the list each time a new element is added to the end of the list, and also decrementing the path length by the distance between fist two point on the list each time beginning point is removed from the list.

Another problem that can be solved by the path management algorithm is the one created by the fact that the orientation of the following robot is constantly changed by orientation of the leading robot. This can be solved by recording the orientation of the leading robot (yaw_L) and saving it as another parameter on the list with its corresponding position. This adjustment changes the nature of this algorithm from being a collection of points into being a collection of positions.

Adjustments that need to be made to the position following algorithm to make it into path following algorithm are simple. The change is required only when it comes to the node from which this algorithm receives information about odometry of the leading robot. Before, odometry information was provided by morbot, er1 or gazebo, now this information is provided by the path management algorithm. Figure 7.2.1. shows changes that are made compared to position following algorithm when compared with Figure 7.1.1.

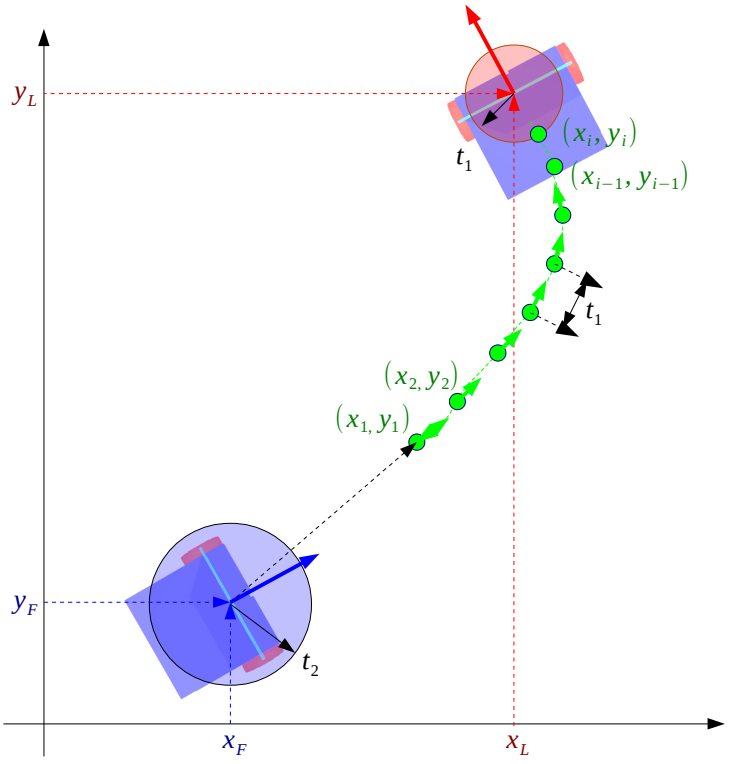


Figure 7.2.1. Leading and Following robot with Path management algorithm

Path management algorithm is not without its faults. The main one being the fact that if the leader robot goes backwards, the follower robot is not capable of following at all. The second problem is the fact that if the following robot passes by the first point on the list, it may not be able to continue following the path at all. This can be treated but at a cost. The parameter for determining the removal of a point from the list, **t2**, can be increased while the parameter for adding a point to the list, **t1**, can be decreased. This allows some flexibility to the path management algorithm, but it will produce noise in the controlling parameters that will not be fixed by the CO filter.

All follower algorithms were specified in **Platoon Algorithms** package under **odometry_follower_test_with_navigation** node. Path management algorithm was specified in **Platoon Algorithms** package under **path_creation_algorithm_with_navigation**

8. GPS Following Algorithm

Unfortunately, this part of the project was abandoned. The reasons for this mainly revolve around the fact that Whycon based localization is not compatible with GPS localization in the sense that limits of these algorithms do not allow them to work together in a meaningful way.

GPS, or more specifically SPS systems provide us with information on position that currently claims 4 meter RMS (7.8 meter 95% Confidence Interval) horizontal accuracy. Vertical accuracy is worse. Some devices/locations reliably can get 3 meter accuracy. Technical document on that specification can be found here [7]. However this is the minimum distance that can be told apart. Whycon and odometry information is much more precise and the control laws used require this information to be like this. Introducing GPS information will simply not increase accuracy of the algorithm.

Another issue with GPS is the fact that even if robots used to realize this project were much bigger and sensitivity of algorithms was much smaller, they would simply not satisfy the basic premise that I am trying to use low-cost sensors for this project. They will still have to be very expansive devices used in order to provide any meaning full data.

Final issue, which is least relevant perhaps, is that the algorithm I used for combining information from odometry, whycon and GPS is already developed extended Kalman Filter. This EKF is simply not designed yet to take information from both visual odometry and GPS [27]. This will however change in the future.

9. Combining Algorithm

Odometry information, as provided by Morbot, ER1 and Gazebo packages, rely only on wheel odometry. This means that position and orientation of robots is calculated only from wheel movement. This inevitably leads to miscalculation as uncertainty of the pose of robots increases over time. This also leads to covariance growing without bonds. This is why it is imperative that odometry of a robot is not only calculated from integration of wheel movement, but should also include information that is more based in reality. Luckily, we have developed a relative localization system which can provide us with such information. The question now becomes of how to combine localization parameters from wheel odometry and whycon odometry. This was done using **Extended Kalman Filter**.

9.1. Extended Kalman Filter (EKF)

Extended Kalman Filter [5] used in this project has been provided as ROS package **Robot Pose EKF** [28]. Robot Pose EKF package is used to estimate the 3D pose of a robot based on partial pose measurements coming from different sources. The idea is to combine measurements from different sensors in order to create good estimation of robot position and orientation. For the purposes of this project only wheel odometry and visual odometry measurements were used to create combined odometry topic.

How dose it work?

Both odometry sources send information to the filter node which computes relative pose differences of each sensor to update the extended Kalman filter. This allows sensor sources to have their own world reference frames, and each of these world reference frames can drift arbitrary over time. Another advantage is the fact that not all sources need to be available all the time. The sources can operate at different rates and with different latencies. If a source reappears over time, the node will automatically detect and use all available sources. This sort of setup allowed maximum independence for each sensor source, which is exactly what my system needed.

One of the obvious indications this made was that localization provided by whycon algorithm can work in relative world reference frame while wheel odometry can have its own, so no changes are need for whycon provided odometry. Another important thing to consider is that if patterns are outside of

camera view and whycon is not providing any information, the robot can simply continue to function with wheel odometry as a sole sensor.

9.2. Covariance

Since covariance and uncertainty of robot pose in a world reference frames will grow, it is not useful to publish this information to robot pose EKF. Instead, sensor sources should publish how covariance changes over time (covariance is published in odometry message). Some manipulation of covariance was required in order to match information from both algorithms. What became apparent in simulations, was the fact that if covariances from whycon and wheel odometry were overly mismatched, robot pose EKF would not give good results once patterns would go out of the camera frame. Even worst results were given once whycon could send out information again. Because of these reasons, covariances were matched so that whycon and wheel odometry would have same reliability to robot pose EKF.

9.3. Combined Odometry Topic

Combined odometry was not used for both leader and follower robots. Wheel odometry was of course provided for both robots. Whycon odometry on the other hand, even thou it was possible to use both *"follower relative to leader"* and *"leader relative to follower"* information, this would have been meaningless. Since whycon odometry is one based on relative positions of one robot to another, it would have been a simple duplication of same information, and would not have shown any improvement on the combined odometry. This claim was derived from observations made during simulations. This now meant that I will be using wheel odometry for one of the robots and combined odometry for the other one. Both possibilities were tested in order to perhaps observe some advantage between them, but no observable differences were noticed between them. They both worked equally good.

Manipulation of covariances were specified in **Platoon Algorithms** package under **navigation** node.

10. Results of Simulations

In order to properly show results of my work, a test was designed which will show exactly how platooning system created works and what are its merits and its faults. Both position following algorithm and path following algorithm were tested. Results of the test made in the simulation are shown in Figure 10.1.1. and Figure 10.1.2..

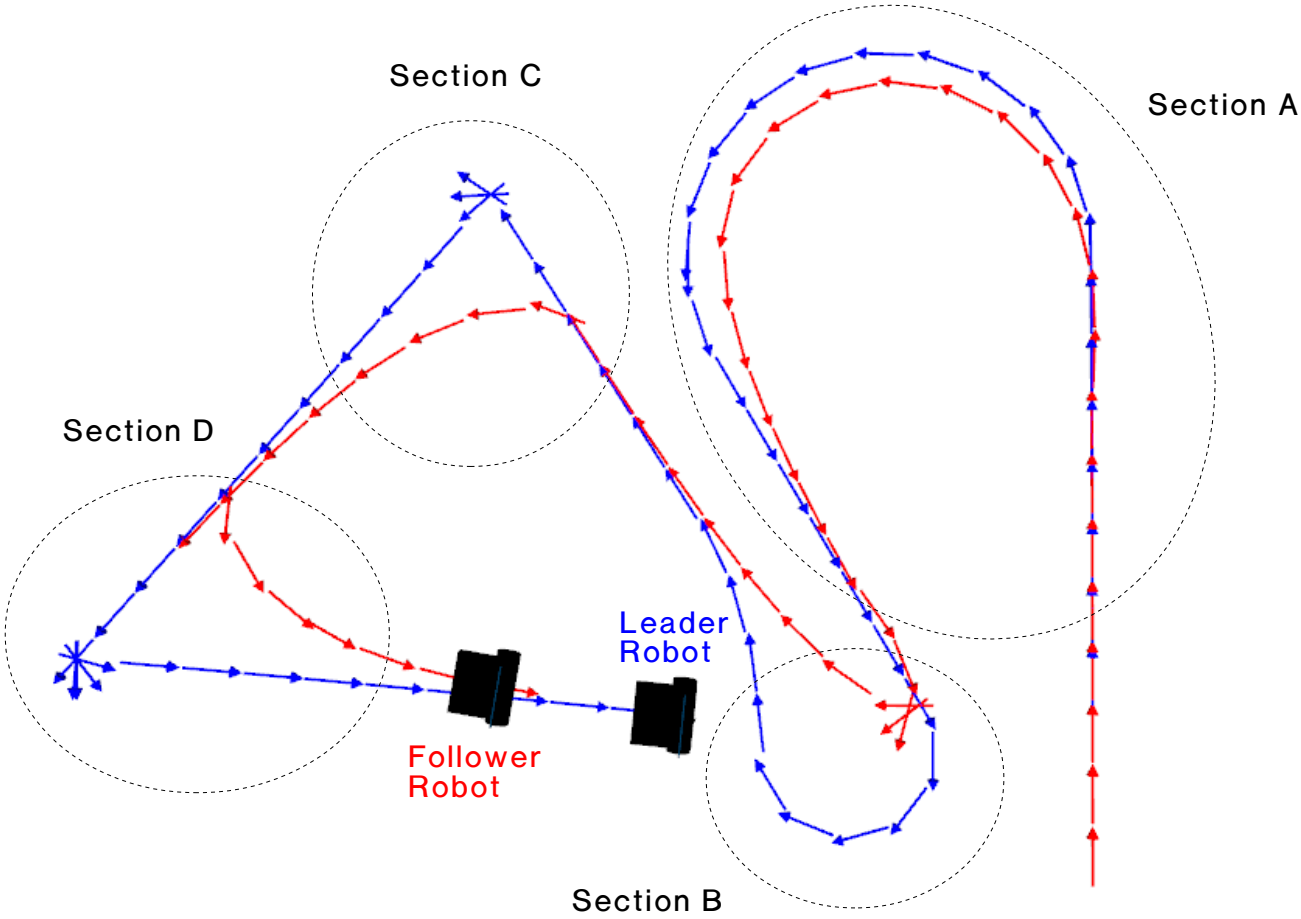


Figure 10.1.1. Simulation of Position Following Platooning System

Simulation test, as It can be observed in (Figure 10.1.1.) is composed of four sections. Section A is platooning system operating in the most ideal way. Both whycon odometry and wheel odometry is available to robot pose EKF. Section B is platooning system functioning without whycon odometry being available as patterns are not out of the camera frame. It is important to note that robot pose EKF continues to function properly without problems. In sections C and D leading robot makes very sharp turns, and similarly to the situation in section B, whycon odometry is not available. However, the position following algorithm shows

robustness as the following robot keeps doing exactly what is predicted.

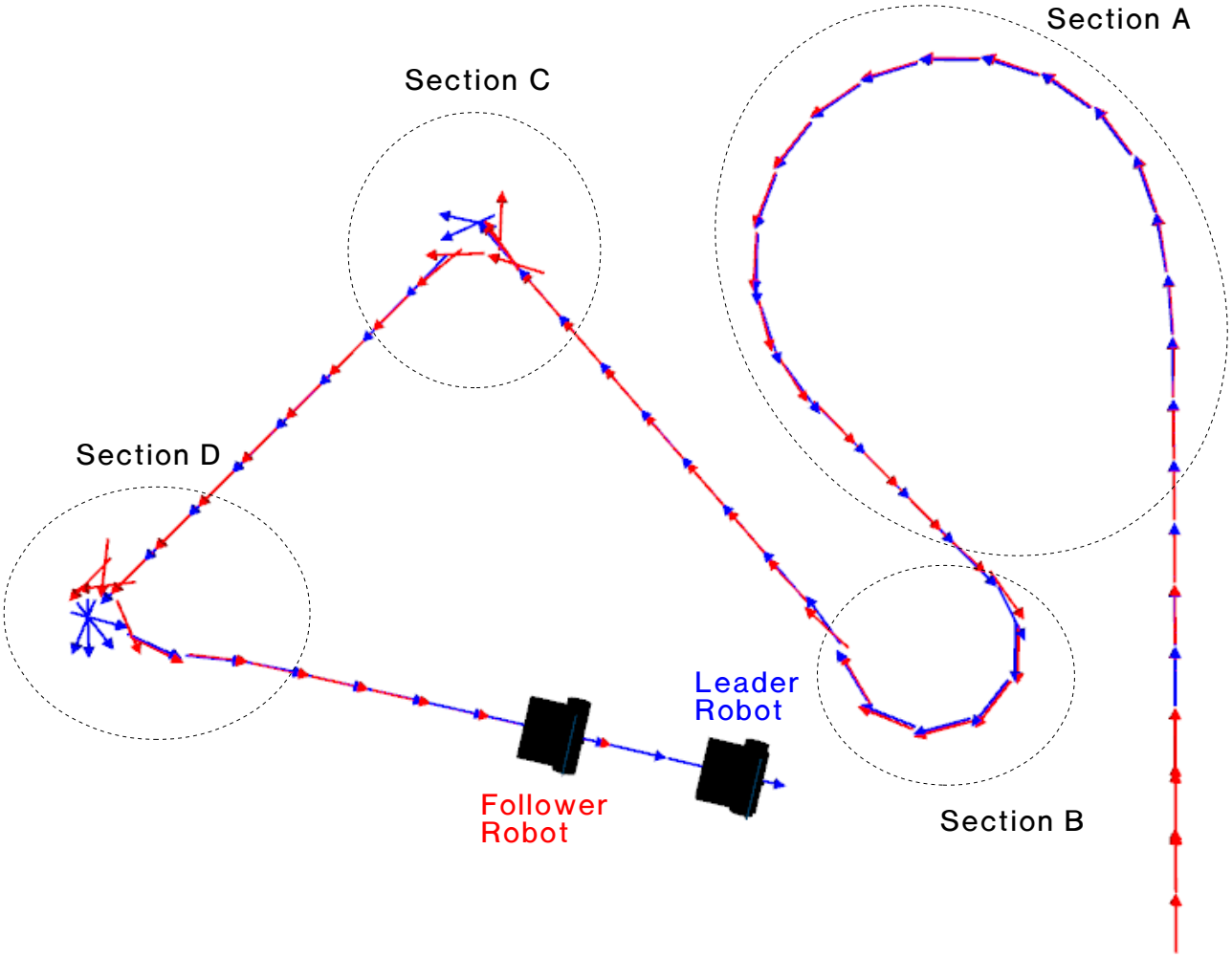


Figure 10.1.2. Simulation of Path Following Platooning System

When comparing results shown in Figure 10.1.1. and Figure 10.1.2. It can be observed that a lot more faithful path was made by the follower as a result of using the path management algorithm. This most apparent in Sections A and B. However, these results also show us that path following algorithm is not as robust as position following algorithm when it comes to sharp corners in sections C and D. Even thou the path is followed faithfully, potential hazards can be observed in these sections.

Visualization of models and their odometries was made using a ROS tool called RViz.

11. Conclusion

With the exception of creating GPS based path following algorithm, all goals of this thesis were fulfilled. A platooning system was created and tested.

Relative localization system with a camera and a pattern detector algorithm, called Whycon algorithm, was implemented in order to determine the position of the follower vehicle relative to the leader vehicle, as well as to determine the position of leading vehicle relative to the follower vehicle.

A path following algorithm based on wheel odometry of both leading and following vehicle was developed and combined with control laws based on parametrized Archimedean spiral and a PID controller. This path following algorithm did show some faults regarding proper management of information provided to controllers. However, these were very specific conditions at which the path following algorithm failed to deliver its full potential. It also showed precision once it was allowed to utilize combined information from Whycon odometry and wheel odometry.

A combining algorithm, based on Extended Kalman Filter, was implemented and developed to a highly satisfactory level and since this thesis is basically investigation of what can be achieved when cheap and weak sensors are combined together, implementation of EKF might be the most important part of this thesis. Combination of Whycon provided odometry and wheel provided odometry worked perfectly in the since that these two sources of information were able to remedy each others weaknesses.

Flexibility and versatility of Robot Operating System has been made apparent. The ease with which one can utilize and create upon other peoples good work is only matched with other peoples good will to help each other in creation of good work.

12. References / Bibliography

[1] Bergenheim C. Shladover S. and Coelingh E. ***Overview of platooning systems*** (2012) Proceedings of the 19th ITS World Congress

[2] Alan Ali, Gaetan Garcia and Philippe Martinet. ***Safe Platooning in the Event of Communication Loss using the Flatbed Tow Truck Model*** (2014)

[3] Tomas Krajnik, Matias Nitsche, Jan Faigl, Tom Duckett, Marta Mejail and Libor Preucil. ***External Localization System for Mobile Robotics*** (2013)

[4] Jong Jin Park and Benjamin Kuipers. ***A Smooth Control Law for Graceful Motion of Differential Wheeled Mobile Robots in 2D Environment***

[5] Sebastian Thrun, Wolfram Burgard and Dieter Fox. ***Probabilistic Robotics*** (2000)

[6] J. G. Ziegler and N. B. Nichols. ***Optimum Settings for Automatic Controllers*** (1942)

[7] John G. Grimes. ***Global positioning system Standard positioning service Performance standard*** (2008)

[8] ***ROS Messages***, *ROS tutorials* (2016) (online) http://wiki.ros.org/msg

[9] ***ROS Topics***, *ROS tutorials* (2014) (online) http://wiki.ros.org/Topics

[10] ***ROS Nodes***, *ROS tutorials* (2012) (online) http://wiki.ros.org/Nodes

[11] ***Gazebo*** (online) http://gazebosim.org

[12] Ioan Sucan. ***URDF***, *ROS package* (2014) (online) http://wiki.ros.org/urdf

[13] Stuart Glaser, William Woodall and Robert Haschke. ***Xacro***, *ROS package* (2016) (online) http://wiki.ros.org/xacro

[14] John Hsu. ***Gazebo Plugins***, *ROS package* (2015) (online) http://wiki.ros.org/gazebo_plugins

[15] Wim Meeussen and Mathias Ludtke. ***Controller Manager***, *ROS package*

(2016) (online) http://wiki.ros.org/controller_manager

[16] Wim Meeussen. ***Robot State Publisher**, ROS package* (2016) (online) http://wiki.ros.org/robot_state_publisher

[17] Mike Purvis. ***Teleop Twist Joy**, ROS package* (2015) (online) http://wiki.ros.org/teleop_twist_joy

[18] ***Writing a Teleop Node**, ROS tutorials* (2016) (online) http://wiki.ros.org/joy/Tutorials/WritingTeleopNode

[19] Ken Tossell. ***UVC Camera**, ROS package* (2015) (online) http://wiki.ros.org/uvc_camera

[20] ***Using ROS Indigo/Jade with a web-cam by the uvc_camera (USB Video Class) package*** (2014) (online) https://defendtheplanet.net/2014/11/05/using-ros-indigo-webcam-by-the-uvc_camera-usb-video-class-package/

[21] ***Blender*** (online) https://www.blender.org/

[22] ***MeshLab*** (online) http://www.meshlab.net/

[23] Melonee Wise, Sachin Chitta and John Hsu. ***Control Toolbox**, ROS package* (2013) (online) http://wiki.ros.org/control_toolbox

[24] Andy Zelenak and Paul Bouchier. ***PID controller Node**, ROS package* (online) http://wiki.ros.org/pid

[25] Control Guru. ***Signal Filters and the PID with Controller Output Filter Algorithm*** (online) http://controlguru.com/pid-with-controller-output-co-filter/

[26] Patrick Mihelich, Kurt Konolige and Jeremy Leibs. ***Image Proc**, ROS package* (2015) (online) http://wiki.ros.org/image_proc

[27] ***Robot Pose EKF, GPS Sensor**, ROS tutorials* (2011) (online) http://wiki.ros.org/robot_pose_ekf/Tutorials/AddingGpsSensor

[28] Wim Meeussen. ***Robot Pose EKF**, ROS package* (2012) (online) http://wiki.ros.org/robot_pose_ekf

13. Appendix: CD Contents

Root Directories	Description
Thesis	PDF format of the Bachelor Thesis
Platoon	Platoon source codes implemented in ROS
Video	Videos from Position and Path follower algorithm implementation