FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Motion Tracking for an Autonomous Race-Car

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Bachelor of Science**

in

**Technical Informatics**

by

**Binder Benjamin**
Registration Number 1226121

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Bader Markus Univ.Ass. Dipl.-Ing. Dr.techn.

Vienna, 1st January, 2001

| Binder Benjamin | Bader Markus |
|---|---|

# Erklärung zur Verfassung der Arbeit

Binder Benjamin
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Jänner 2001

Binder Benjamin

# Kurzfassung

Diese Arbeit beschäftigt sich mit dem Positions-Tracking von Fahrzeugen. Hierbei muss die Position von internen Sensoren wie Gyroskopen, Beschleunigungssensoren und Umdrehungszählern abgeleitet werden. Dazu muss ein Modell zur Positionsbestimmung, beziehungsweise eine Methode zur Fehlerabschätzung implementiert werden. Für diese Anwendug gibt es dabei zwei verschiedene Modelle. Das „Odometry Model“, bei dem die Position anhand von Lenkwinkel und Bewegungsgeschwindigkeit abgeleitet wird und das „Velocity Model“, welches die Daten anhand von Translations- und Rotationsgeschwindigkeit abgeleitet wird.
Um die Realisierung eines solchen Modells zu ermöglichen, muss das Fahrzeug auch dementsprechend vorbereitet werden. In dieser Arbeit wird die Inbetriebnahme einer IMU (Inertial Measurement Unit), welche für das „Velocity Model“ verwendet werden kann, und eines „Odometry Models“, welches von den Umdrehungsdaten eines Brushless Motors abgeleitet wird. Weiters wird die Funktionsweise und Programmierung eines Brushlesscontrollers beschrieben. Die Kontrolle und Überwachung des Fahrzeugs wurde mittels eines ArduinoUNOs und eines RaspberryPI's realisiert.
Diese Kombination von Sensoren, Aktoren und CPU bringt einige Vorteile. Durch die Verwendung zweier Prozessoren können die erforderlichen Aufgaben für Tracking und eine darauf aufbauende Lokalisierung sehr gut getrennt werden. Weiters ist eine IMU für die Verwendung eines „Velocity Models“ und ein Encoder für die Realisierung eines „Odometry Models“ vorhanden. Für den Encoder des Modells werden die internen Sensoren des Brushless Motors verwendet um die Anzahl der verwendeten Komponenten gering zu halten.
Um diesen Ansatz zu testen wurde ein Tamiya RC-Auto umgebaut. Es wurde ein Brushless Motor verwendet, welcher durch einen selbst programmierten Controller angesteuert wird. Weiters wurde das „Odometry Model“ auf einem ArduinoUNO implementiert und mittels Raspberry PI und ROS (Robot operating System) ausgewertet. Um die Integration eines „Velocity Models“ zu ermöglichen wurde des weiteren eine IMU in Betrieb genommen.

# Abstract

This bachelor thesis is about position-tracking of vehicles. Therefore the position has to be derived from internal sensors like gyroscopes, accelerometers and wheel encoders. A model for position determination and error estimation has to be implemented. There are basically two different models to fulfil this task. On the one hand side is the odometry model, which determines the position from linear velocity and the steering angle of the car. At the other hand side is the velocity model which describes the position using the rotational and translational velocities of the car.
To prepare for this task several components have to be added to such a vehicle. For determining the position from velocities a IMU (inertial measurement unit) can be used. To determine the position via the odometry model wheel encoders are needed. One can use a brushless motor to control the car and get the cars speed. In this thesis the implementation of such a controller is described. Furthermore the usage of a Raspberry PI and a Arduino UNO for controlling the car is described.
Because two CPU's are used the tasks for tracking and later for localization of the car can be split clearly. To prepare the car for two different motion models a IMU and a wheel encoder are implemented. Because a brushless motor is used the wheel encoder can be replaced with the internal sensors of the brushless motor.
For testing this approach a Tamiya RC-car is rebuilt. The motor of the car is replaced with a brushless Motor and a self programmed brushless controller. Furthermore a odometry model is implemented on a Arduino Uno. For integration of the velocity model a Pololu AltIMU-10 v4 is used. To control the car a Rapberry PI with ROS is used.

# Contents

CHAPTER 1

# Introduction

For many tasks in mobile robotics a position tracking strategy is needed. In this Bachelor Thesis such a strategy for vehicles with ackerman drive is presented. Such a position tracking model derives the vehicles position from internal sensors like gyroscopes, accelerometers and wheel encoders. Furthermore the model estimates the error of the actual position. Basically there are two motion models to choose from. The first one is the odometry model which derives the position from speed, captured by wheel encoders and the steering angle. The second one is the velocity model which uses gyroscopes and accelerometers to calculate the actual position from translational and rotational speed. For each of these two models different sensors have to be chosen. To move the car a motor has to be chosen and controlled from a CPU. Furthermore a control structure of the whole system has to be built up. Including the choice of CPU's, additional sensors and a operating system.

To fulfil this task the choice of a motor is discussed. Additionally a brief explanation of the chosen brushless motor is given. For controlling the motor the principle of a brushless controller is explained and implemented. The choice of the motion model is also presented in this paper. Sensors for both models are explained. For the velocity model a IMU is used to derive the rotational and translational speed from the IMU's acceleration and rotation values. For the odometry model the internal sensors of the brushless motor are used to get the rotational speed of the motor and calculate the speed. For the steering angle the known position of the steering servo is taken. To built up the control structure of the car its tasks are split into low level and high level ones. Therefore two CPU's are used to process the different tasks.

The described approach is used for some reasons. To choose the motor there are mainly two different types to use as DC motor. The BLDC (brushless DC motor) and the brushed DC motor. The BLDC motor is chosen because it has many advantages compared to the DC motor. Enhanced speed, a good dynamic response, longer operating life, noiseless operation, high speed ranges etc. are few to mention[SJ14]. To build up a flexible base frame for other applications sensors for both motion models are prepared.

The benefit of the velocity model is the absence of uncertainties like the slip of the wheels and bumps. Whereas the odometry model tend to be more accurate because a better accuracy of the sensors. To increase the accuracy of the used model, both models can be combined. As mentioned above the tasks of the car are split into low and high level tasks. These low level tasks are the estimation of the car's position, the communication to the sensors over their specific interfaces and the control of the motor. High level tasks are localization of the vehicle and visualization of the captured data. The choice of two processors for the control structure is because the mentioned low level tasks are dependent from the car and the high level tasks are more abstract and independent from the car. So the vehicle and the main application can be exchanged arbitrarily.

To test the presented approach a Tamiya RC-care is used to build up such a car. A brushless motor is used to drive the car. Furthermore a brushless controller is implemented. Therefore a Arduino UNO with three Half-Bridge drivers is used to generate the needed control signals for the motor. To keep the revolution speed of the motor constant a fuzzy logic[AK14] controller is implemented. Also the IMU is connected to the Arduino and the data is prepared for high level applications. For the odometry model the revolution speed is taken from the internal sensors of the brushless controller. The model is calculated afterwords on the Arduino and sent to the main CPU over serial communication. To prepare a flexible interface to other applications ROS (robot operationg system)[Ng09] is used on a Raspberry PI 2. For simulation and visualization of the data Matlab, RVIZ and the razor IMU node are used on another workstation connected over wireless LAN to the Raspberry PI.

CHAPTER 2

# State of the Art

## 2.1 Motion Model

A motion model observes the robots movements and its uncertainties. That means a model is needed, which predicts the next position depending on the last position $p(x_t|x_(t-1), u_t)$ and the vehicles actions. In 3D the position of a vehicle is described by x,y,z and roll, pitch, yaw. Because the car is moving on the floor the position representation can be simplified to x,y and $\theta$. One can find two different motion models for locating moving vehicles. The odometry model, which is used if a system is equipped with wheel encoders and the velocity model for systems without encoders. In the shown race car the odometry- and the velocity-model are prepared. The odometry model uses wheel encoders to read the vehicles speed and distance. This model tends to be a more accurate than the other one, but there are still inaccuracies left like different wheel diameters, bumps, slippery ground and so on. The velocity model has the advantage of bypassing such inaccuracies. So one can use these two models to combine them for example with a Kalman filter to improve accuracy. To describe the noise of the model a variance matrix is introduced which is used by different filters to model the growing inaccuracy of the pose prediction.

## 2.2 IMU

A typical 6 degree of freedom inertial measurement unit usually consists of three accelerometers and gyroscopes.[Gam01] From the data of these elements, the vehicles velocity vector and attitude can be determined. To calculate the attitude of the IMU, the gyroscopes in addition with the accelerometers have to be used. The gyroscope has an output of degrees per second. These output tend to drift over time. The accelerometer describes accelerations without any drift but is really unstable over short periods. Thus the outputs of the sensors has to be combined. To calculate the angle from the accelerometers the gravity is used. This gets inaccurate through movements of the car. But due to the fact that the accelerations of the car are very small compared to gravity and a filter is used to compensate them the calculation works really fine. Normally a complementary- or a Kalman-filter is used, to combine the last state and the two measurements, to get a correct value. After the attitude is found the acceleration in x and y direction can be determined.

## 2.3 Brushless DC Motors

A brushless DC Motor (BLDC-motor) consists of rotor and stator with windings. The basic structure of the motor is shown in Figure 2.3. A big advantage compared to the DC-motor is the long lifetime of the BLDC-motor because the absence of brushes. The stationary stator windings are usually three phases, which means that three separate voltages are supplied to the three different sets of windings. As one can see three sinus signals with 120 degree phase offset can be used to control the motor, shown in Figure 2.3. This operation mode is called gimbal mode. This motors are used instead of stepper motors to hold at a specific position. The position is described of the specific combination of the sinus signals. Not every motor can be used in this mode because many windings are needed to fulfil the responsibilities for a Gimbal motor.
For controlling the motor in a faster and more dynamic way another strategy can be used. Therefore every sine-wave is split into rectangular signals to generate the voltage curve. In Figure 2.3 a sine-wave, the control signal and the real signal is shown. It can be seen that the windings inductivity reforms the control signal into a trapezoidal signal. In this case the position where to switch the rectangles has to be known. This can be done at one hand by measuring the zero crossings of the motor current or at the other hand by using Hall sensors. Because the current and so the magnetic field are lagging, the zero crossing of the real signal is normally to late. To get rid of this lagging every commutation of the windings is done earlier. This angle correction is called timing. A soft timing means small angles and leads to an higher efficiency. The opposite is a hard timing which leads to higher accelerations and less efficiency. [BÃij13]
Starting sensorless motors is problematic, because the position at the beginning is not known. The first strategy is to power exactly one winding to bring the rotor in the right starting position. The disadvantage of this method is a uncontrolled acceleration behavior in the first step. The second method is to start at a arbitrary position with low frequency and increase the frequency slowly. The problem therefor is, that the Magnetic field can loose the stator an the motor gets out of step. Because of these starting problems brushless motors where first used in model aircrafts and motorboats because there are less load changes while running the motor. To get rid of these problems one can use the hall sensors to accelerate the motor. Because in this case the position is known every time and the controller can set the right signals from the beginning for a smooth acceleration.

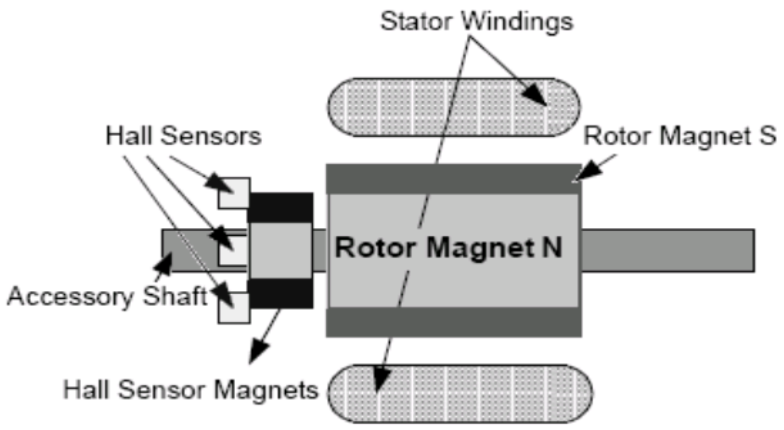


Figure 2.1: Basic structure of the BLDC motor[AK14]

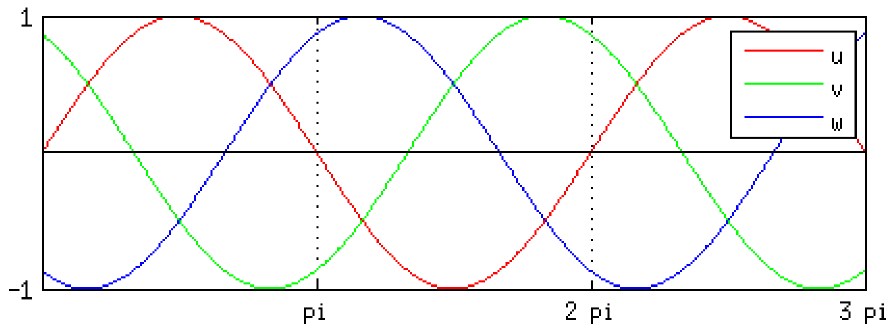


Figure 2.2: Control signals of a brushless gimbal motor

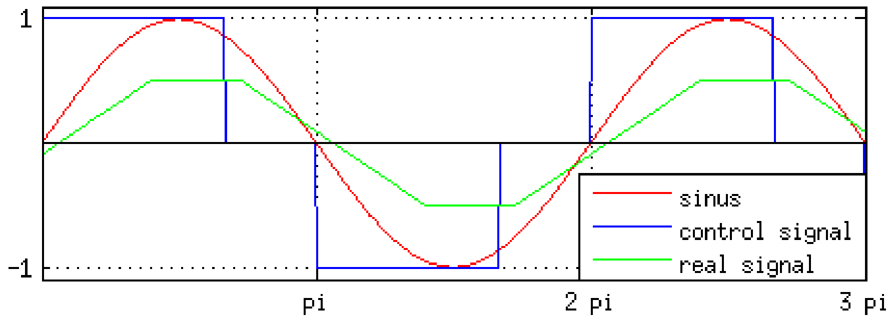


Figure 2.3: Control signal of a brushless controller

CHAPTER 3

# Method

This chapter is split in two main parts. The first part is about the basic structure for Hardware and Software. It shows the decisions whether a servo with a brushless controller or a programmable micro-controller should be used. Furthermore the decision of the motor is described and the advantages and disadvantages of the motors are explained. The second part gives a brief description of the used micro-controller, the brushless controller, the integration of the brushless controller in the IMU program and the communication to the terminal. Moreover a brief explanation of the terminal node for communication with the micro-controller is given.

## 3.1 Basic Structure

To prepare a clear structure, the cars tasks are split into low and high level tasks. Low level tasks are controlling the signals of the brushless motor, reading the IMU over the $I^2C$ interface and position tracking. High level tasks are for example visualization and localization. As one can see low level tasks are dependent to the car and the used components while high level ones are more abstract. To present a flexible interface two CPU's are used to quickly exchange the vehicle or the main application. A well known framework to program such flexible interfaces is ROS (Robot operating system). This framework consists of the roscore which serves as master-node and a arbitrary number of other nodes which can communicate over messages. So future tasks can subscribe to the control and response messages of the robot and interpret them. A detailed description for ROS is shown in the paper [Ng09]. For running ROS a debian operating system is needed. Therefore a Raspberry PI is used because it's small in size and is powerful enough to run debian. Additionally a wireless connection can be established to provide a connection to other workstations. To process low level tasks, another CPU is needed which provides interfaces for the used components and is fast and real time capable. In this case two methods where tested. The first one was to use a MicroMaestro servo controller and

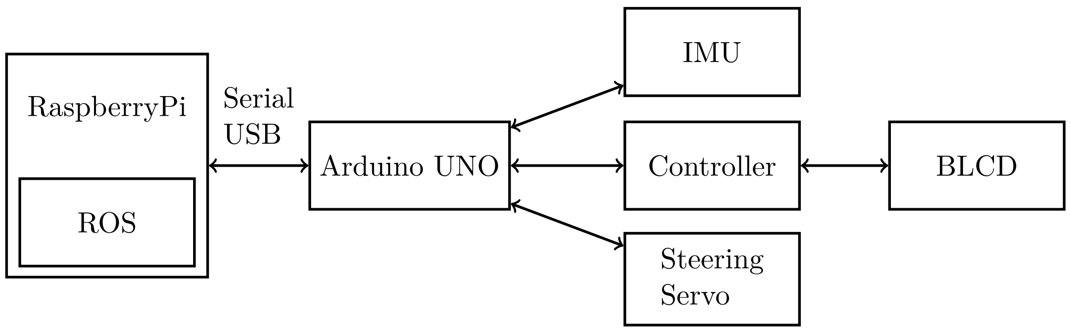


Figure 3.1: Basic structure of the race car

additional hardware to control the car. The second one was to use an ArduinoUNO to process the data. Furthermore the low level sensors an actors have to be chosen to fit into the given control structure. In Figure 3.1 the basic structure of the car is shown.

### 3.1.1 High Level CPU (Raspberry PI)

As mentioned above a Raspberry PI with ROS is set up for the Main application. The supported version of the Raspberry operating systems for ROS is Raspbean because it's based on debian. Additionally a W-LAN stick is installed on the raspberry PI to grant access from other workstations. For communicating with the Arduino a Terminal node has to be created. The communication to the low level CPU works over USB serial interface with the boost library. Because the used IMU comes with a working program the control messages for the other components are defined in a similar way to include them into the program. To read the car's status and control it, three ROS nodes are used. The Razor IMU node to read the IMU data an visualize it, the ROS joy node to control the car via game pad and rviz to visualize the position of the car. To link the captured data to the right position on the car the ROS tf package is used.

### 3.1.2 Sensor Level CPU (Arduino UNO)

The sensor Level CPU is responsible for controlling the car, reading sensors and presenting the data in a useful way. Traditionally RC-cares where powered from fuel engines. The carburetor of these fuel engines was controlled with a small servo motor. To make it easy to switch from fuel engines to brushless or DC motors the controllers of these motors are also controlled by servo values. So the first idea was to use the MicroMaestro for controlling the motor-controller and the steering of the car. The benefit of using the MicroMaestro was the tight structure and the quick and easy implementation of it. There is also no need to program a Motor controller, because the original one from the RC-car can be used. But there are less possibilities to integrate sensors in this interface. Additionally the motor controller is optimized for high accelerations and maximum speed, but this car has to be able to move slowly to explore it's ambient and there is no possibility to alter the controller in this way. Additionally there is no speed response from the motor and odometry sensors have to be added. So the next idea was to test an Arduino UNO with a motor shield to integrate the controller and the sensors in the low level CPU. The biggest benefit of this solution is that the Arduino is a flexible and easy programmable Microcontroller. At the other Hand a fully working motor controller has to be programmed in C, a communication interface has to be implemented, and knowledge of the used sensors is needed. Which is much more work than the first solution. But using the MicroMaestro limits the possibilities too much for this application. Especially when adding new sensors it causes many problems. So the Arduino UNO is used for testing the car.

### 3.1.3 Sensors and Actors

To control the car a steering and a motor has to be choosen. For the steering control of the car a servo motor is taken, which is controllable by a PWM signal. The most common motors are a DC motor and a brushless-motor. The brushless DC Motor has many advantages compared to the DC motor. It has a good dynamic response, a longer operating life, less noise and a higher efficiency. Because of these advantages and the fact that BLDC motors are slowly replacing DC motor as well as there is no need for additional odometry sensors, the decision was taken to use a BLDC motor. Finally sensors have been chosen, to measure the moved distance. The simplest choice for such sensors is to use odometry sensors, because wheel encoders are really easy to read and very cheep. If a brushless-motor is used the distance can be determined without using any additional sensor. Another cheep but more challenging method is to use an IMU for a velocity based sensor model. The odometry sensors tend to be more accurate but the IMU is better for harsh environments because odometry sensors can't compensate problems like wheel slip and bumps. To get the best of both for example a Kalman-filter can be used to combine both measurements. So a IMU and a odometry measurement method are prepared for the localization task, because both of them are cheep to implement and the car is prepared for many different tasks.

## 3.2 Components

### 3.2.1 Low-Level CPU

A Arduino UNO (Figure 3.2.1) is used to implement the sensor level CPU, because Arduino offers many libraries for programming AVR micro-controllers and a active community. But most of the programming work is done in C++ and not in the Arduino environment, to present new flexible Arduino libraries. Additionally this design decision makes the code more readable and easier to use. For the IMU integration the Razor IMU ROS node is used, which serves a Arduino program to communicate with the IMU. Additionally a new serial interface library for Arduino is programmed. Furthermore the calculation for the odometry-model is integrated into the program.

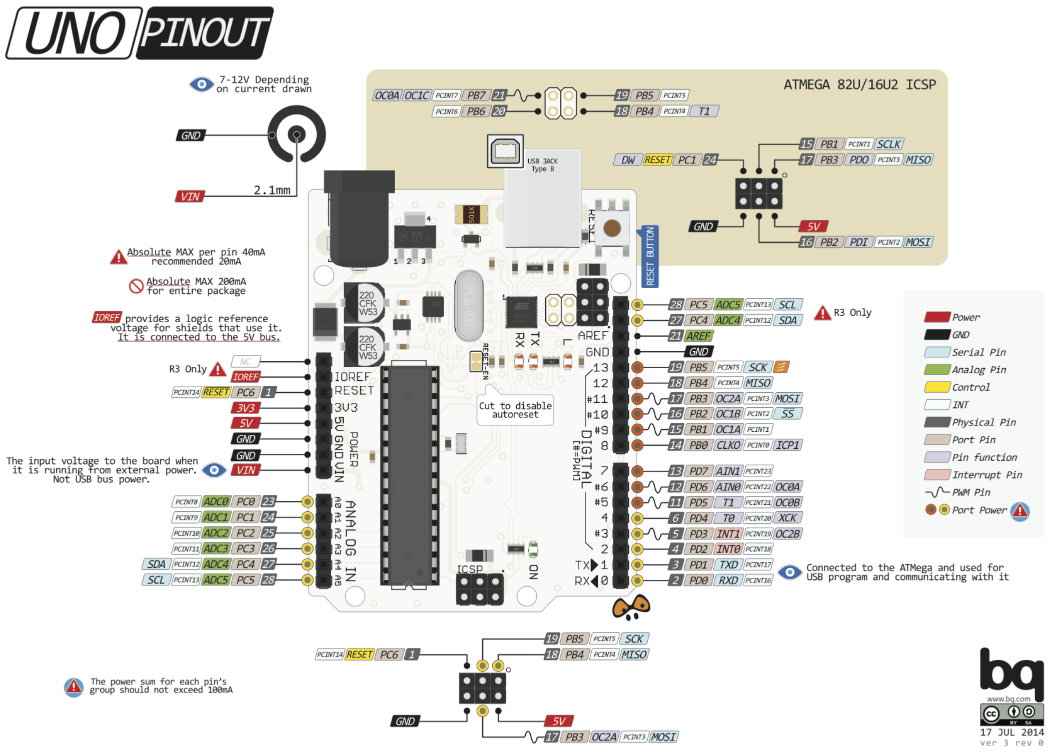

UNO PINOUT
7-12V Depending on current drawn
GND
VIN
2.1mm
ATMEGA 82U/16U2 ICSP
Absolute MAX per pin 40mA recommended 20mA
Absolute MAX 200mA for entire package
IOREF provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
R3 Only
The input voltage to the board when it is running from external power. Not USB bus power.
Cut to disable autoreset
Connected to the ATMega and used for USB program and communicating with it
The power sum for each pin's group should not exceed 100mA
Power
GND
Serial Pin
Analog Pin
Control
INT
Physical Pin
Port Pin
Pin function
Interrupt Pin
PWM Pin
Port Power
17 JUL 2014
ver 3 rev 0

Figure 3.2: Arduino UNO

### 3.2.2 Motion Controller

For the brushless motor controller a brushless motor shield from LXRobotics is used to control the motor. It consists of three FAN7093 High-Current PN Half-Bridge Driver to generate the signals for the motor. Furthermore three inputs are used for detecting the rotor state. The used inputs and outputs on the ArduinoUNO are shown in Figure 3.2.2 and Figure 3.2.2. Basically a state table is generated to match the rotor position with the actual state (Figure 3.2.2). On every pin change of the three inputs, the state is determined and the next state is set. Therefore the next state is set for driving forward and the previous is set for driving backward. Additionally the described timing of section 2.2 is implemented. To determine the position $\alpha$ degrees before the next step a pose prediction is implemented. This prediction uses the moving average of the last N step-change timings.
For driving the motor at different speed values the voltage has to be changed. Normally this is done by using a pulse width modulation which is smoothed by the inductivity of the windings. Therefore a Timer is used in CTC mode. The CTC mode generates an Interrupt on 2 different timer matches, where the timer gets reseted on the higher one. Now the actual state of the driver is set on reset of the timer and reseted on the first

| Phase | AVR Pin | Arduino PIN |
|---|---|---|
| IN1 | PC0 | 23 |
| IN2 | PC1 | 24 |
| IN3 | PC2 | 25 |

Figure 3.3: Input PIN Assignment Brushless Shield

| Phase | AVR Pin | Arduino PIN |
|---|---|---|
| U | PD2 | 4 |
| U_INH | PD3 | 5 |
| V | PD4 | 6 |
| V_INH | PD5 | 11 |
| W | PB0 | 14 |
| W_INH | PD7 | 13 |

Figure 3.4: Output PIN Assignment Brushless Shield

compare match, if the angle is smaller than the angle prediction. As one can see the first compare match is to change the voltage level and the second one is to change the PWM frequency of the signal. These PWM frequencies are in between 200Hz and 64kHz for motors with high or low inductivity.
To ensure accurate speed behavior under different loads a controller has to be implemented. For implementing such a controller a PID controller or a fuzzy logic controller can be used. Because the mathematical model of the motor has to be known and adapted for different motors while using a PID controller, a fuzzy logic controller is implemented. The structure of this controller is shown in Figure 3.2.2. The Fuzzifier converts the given value into the fuzzy value which can be used to decide the next step. The Defuzzifier converts the decision into a control value for the BLCD motor. The Inference engine and the Rule base are used to change the output depending on the input. So the voltage can be quickly adapted to load changes and keep the speed constant.
Another advantage of the controller is to be able to start the BLCD motor with it's full torque. After the reference speed is reached the controller decreases the speed-value automatically. To start the motor only the first step has to be set because our brushless motor has hall sensors. In the results sections one can see that this approach is not working really well and has to be improved. Because the motor controller is interrupt driven, it will never recognize if it has stopped unexpected. To get rid of this problem a watchdog-timer is implemented to guaranty a continuous control process.

| State | in_1 | in_2 | in_3 | out_U | out_V | out_W |
|---|---|---|---|---|---|---|
| 1 | high | low | high | low | high | off |
| 2 | low | low | high | off | high | low |
| 3 | low | high | high | high | off | low |
| 4 | low | high | low | high | low | off |
| 5 | high | high | low | off | low | high |
| 6 | high | low | low | low | off | high |

Figure 3.5: Brushless controller state-table

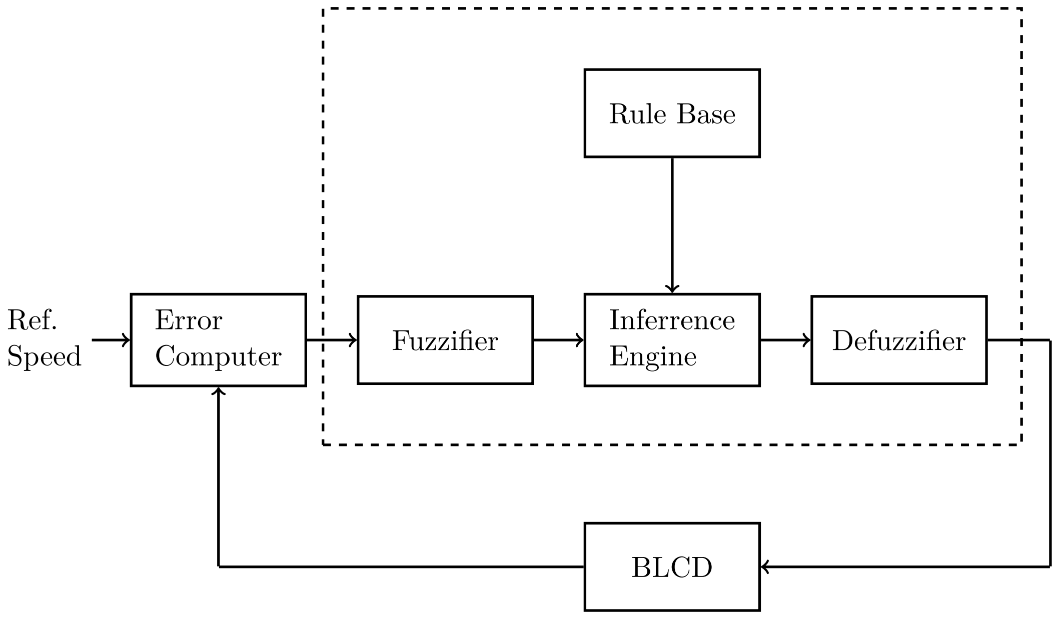


Figure 3.6: Fuzzy logic controller

### 3.2.3 Motion Sensors

For the velocity model a Pololu AltIMU-10 v4 is used. It comes with a working program for ROS and Arduino called Razor IMU. This program uses serial communication over USB and the $I^2C$ interface to communicate to the measurement unit and the ROS environment. The two $I^2C$ pins, are located on PC4 (pin 27) and PC5 (pin 28). The Razor IMU program returns the attitude, the accelerations, the gyroscope- and compass-values from the IMU and displays them. For the odometry model the hall sensor response from the motor and the angle from the steering servo controller are taken to calculate the odometry data. The values are updated by calculating the distance since the last update and adding them to the odometry values. The car's frame is in direction of the car's initial position as well as the IMU frame.

### 3.2.4 Control Interface

Because there is an existing program which works with the Pololu IMU, this program is used and the motor controller and the odometry model are integrated. To keep compatibility to the old Razor IMU Terminal node the existing messages are kept. Two new control and response messages for the BLDC motor and a new odometry message is introduced. The new commands are shown in Figure 3.2.4. For the IMU it looks like #YPRAG [yaw],[pitch],[roll],[acceleration x],[acceleration y],[acceleration z],[gyroscope x],[gyroscope y],[gyroscope z]. As one can see the orientation with roll pitch and yaw, the accelerations and the gyroscope values are given. The new odometry massage is formated as followed #OVD [frameNumber],[linear speed],[angular speed],[odometry x],[odometry y], [odometry $\theta$],[covariance 1,1],[covariance 1,2],[covariance 1,3],[covariance 2,1],[covarianz 2,2],[covarianz 2,3],[covarianz 3,1],[covarianz 3,2],[covarianz 3,3]. One can see that the message consists of the speed and the steering angle as well as the odometry data and its covariances.
To handle the new messages a new ROS Terminal node has to be written. This node basically has to capture the serial data and publish them into ROS. Additionally it has to subscribe to control messages and send commands on the serial interface. The expected ROS messages are geometry_msgs::Twist_data or sensor_msgs::Joy_data and the presented ones are nav_msgs::Odometry and sensor_msgs::Imu. Additionally a stamped tf message is presented to ROS.

| command | function | response |
|---|---|---|
| #vs [speed] | sets the given speed | #v [speed] |
| #vg | returns the actual speed | #v [speed] |
| #as [angle] | sets the new steering angle | #a [angle] |
| #ag | returns the actual angle | #a [angle] |

Figure 3.7: IMU serial commands

CHAPTER 4

# Result

To test the autonomous race car a RC car body from Tamya is used. Additionally a Raspberry PI 2, a GM brushless motor, the LXRobotics brushless shield, a gearbox, a Arduino UNO and a Pololu AltIMU-10 v4 was used. For powering the Race car, the Rapberry and the Arduino a standard 7.2V battery pack is used. Because the car drove to fast while testing finally a gearbox with a transmission of 7.4:1 was used to reduce the cars speed. The overall transmission is 61.2 to 1 with 66 mm wheel diameter.
Another problem with the brushless shield was, that the given response pins for the back EMF where lying on the SPI interface of the Arduino. So the circuit paths where removed and only the Hall sensors where used for positional response on pins PC0 to PC2. For testing everything a Oscilloscope, the Razor IMU node, the ROS-joy node and RVIZ are used.

## 4.1 Motion Controller

The first part was to program a working brushless-motor controller. Therefore the mentioned strategy in the method chapter was used. For testing the angle prediction uses a moving average of 6 updates, and the $\alpha$-value is set to 5 degrees for a soft timing. Because of the limited numbers of timers on Arduino UNO the angle-prediction step and the controlling task where moved into the PWM overflow interrupt to update the values with a constant period. That means that the PWM frequency is not adjustable to lower values, because the angle prediction has to be called fast enough to work in the right way. Compatibility to other brushless motors is mainly given because motors with the need of lower PWM frequencies are also moving slower and so lower frequencies are possible again.
The fuzzy logic controller was implemented in a very simple way. It just adds or removes a specific value to or from the PWM value if the speed difference is in a defined corridor seen in Figure 4.1. Testing these values showed that such a simple controller is enough

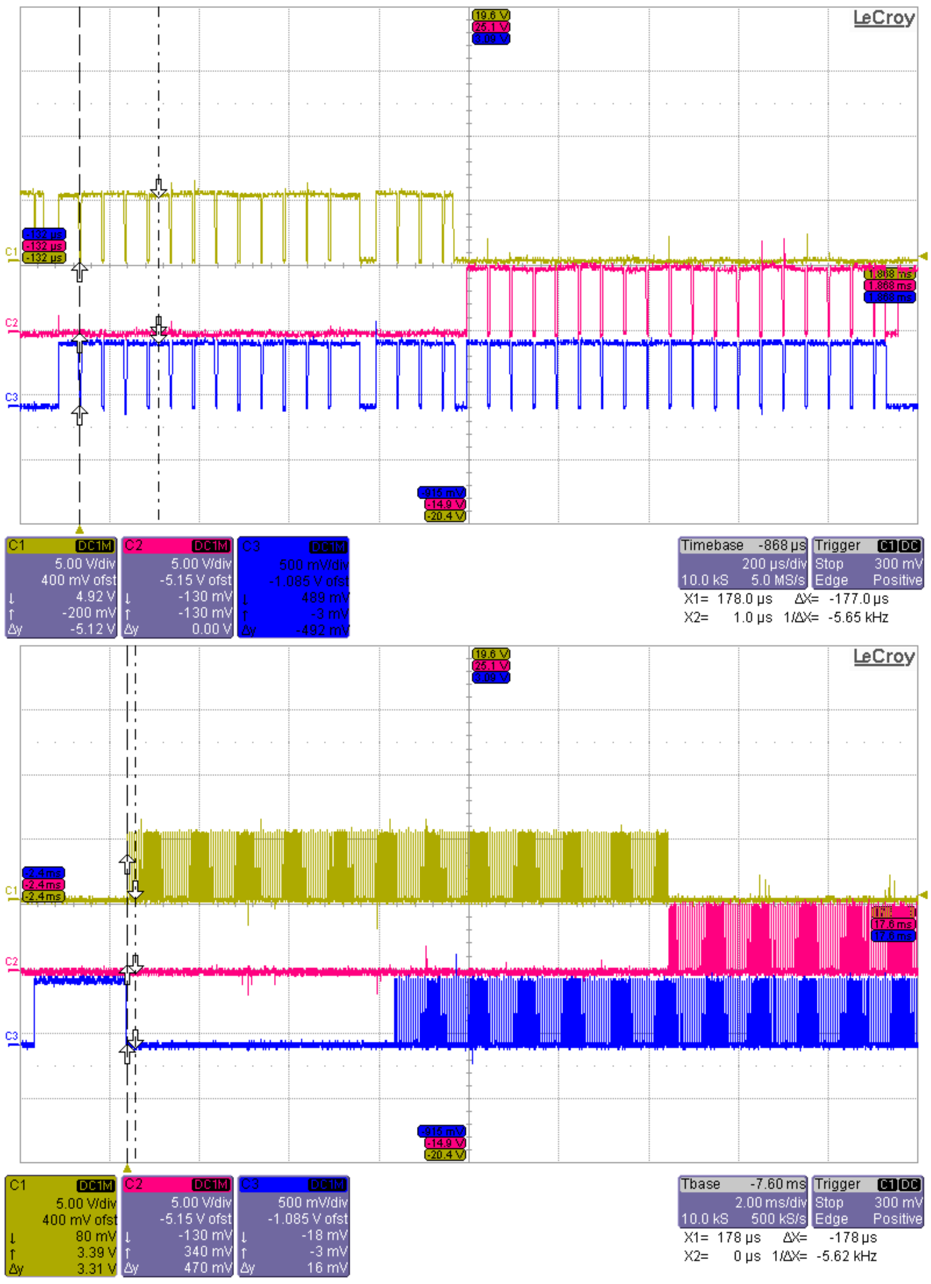
for this task. Like the angle prediction the controller is also dependent from the PWM frequency because if the frequency is higher it has more calls to add values to the PWM value and the controller overshoots the target more then with the lower one and vice versa.
Theoretically it needs only one pulse to start the brushless motor but while testing it turns out that while startup the rotor can loose the magnetic field and stops. One method to get rid of this problem is to detect, if the rotor stops and start it again. Therefore a watchdog-timer is implemented to detect if the pin change interrupts from the hall sensors are appearing constantly. If such a stop is detected the watchdog-timer tries to restart the motor. With this method it takes really long to start the motor and the startup is really irregular. So another start routine was implemented. The motor detects the actual position of the rotor and sets up the next state as usual. But now the controller doesn't wait for the next interrupt on the PCINT pins. It continues increasing the steps with a delay in range of milliseconds until the rotor has found a stable rotation. This rotation is defined as stable if 6 states where correctly changed.
The least speed the brushless driver managed to do was 24.0 turns per second. This is a minimum speed of 0.025 m/s when the transmission and the wheels are taken into account. The maximum speed of the car is about 190 turns per second which is a total speed of 0.2 m/s. Without the gearbox the speed of the car would lie within 0.2 and 1.6 m/s. One can see the timing diagram of the brushless controller's PWM signal in Figure 4.1, which runs at 20kHz. The above picture shows the pulse with modulation pulses for a speed of 180 turns per second. The blue signal is about 1800$\mu s$ long, which is two periods on. So it can be easy seen that the motor turns in fact 180 times per second. In the second picture the PWM signal of a speed of 27 turns per seconds is shown. Therefore it can be seen that the PWM of the yellow signal is 12ms long on.

| speed difference | value to add |
|---|---|
| -0.5 < diff < 0.5 | 0 |
| diff > 0.5 | 1 |
| diff > 30.0 | 5 |
| diff < -0.1 | -1 |
| diff < -30 | -5 |

Figure 4.1: Fuzzy logic rule base

## 4.2 Motion Model

As described above the Razor IMU program is used and changed to the cars needs. To integrate the brushless controller, the main routine has to be changed. The initialize function of the written brushless controller and the command decoding has to be integrated in the main loop. Everything else except setting the speed, works interrupt driven. A big



```
C1  DC1M            C2  DC1M            C3  DC1M
    5.00 V/div          5.00 V/div          500 mV/div
    400 mV ofst         -5.15 V ofst        -1.085 V ofst
↓   4.92 V          ↓   -130 mV         ↓   489 mV
↑   -200 mV         ↑   -130 mV         ↑   -3 mV
Δy  -5.12 V         Δy  0.00 V          Δy  -492 mV

Timebase   -868 µs   Trigger   C1 DC
      200 µs/div     Stop     300 mV
10.0 kS   5.0 MS/s   Edge   Positive
X1=  178.0 µs   ΔX=  -177.0 µs
X2=    1.0 µs   1/ΔX=  -5.65 kHz
```

```
C1  DC1M            C2  DC1M            C3  DC1M
    5.00 V/div          5.00 V/div          500 mV/div
    400 mV ofst         -5.15 V ofst        -1.085 V ofst
↓   80 mV           ↓   -130 mV         ↓   -18 mV
↑   3.39 V          ↑   340 mV          ↑   -3 mV
Δy  3.31 V          Δy  470 mV          Δy  16 mV

Tbase      -7.60 ms   Trigger   C1 DC
       2.00 ms/div    Stop     300 mV
10.0 kS   500 kS/s    Edge   Positive
X1=  178 µs   ΔX=   -178 µs
X2=    0 µs   1/ΔX=  -5.62 kHz
```

Figure 4.2: Timing diagram of the brushless controller. In both pictures the brushless motor's PWM signals of the same situation with different speeds are shown. Additionally there is a input which tells the half bridge driver if the signal is positive ore negative, which is not shown in these pictures. The first two signals (yellow and red) are negative and the blue one is positive.

problem was that the Arduino serial library was to slow for the communication with the motor. To improve the speed, a new serial library with ring buffer is implemented for Arduino. Now the car reacts very quickly to changes in speed and is smoothly controllable over game-pad. To respond to an odometry request a calculation of the actual position is needed. Therefore the distance to the start point of the car is calculated incrementally. We have 6 values for this purpose the $odometry_{x,y,\theta}$ point and the variance to this point $variance_{x,y,\theta}$. In Figure 4.2 the motion model for the Ackerman drive is calculated. Therefore the x, y and theta coordinates can be approximated like seen in Equations one to four. To calculate the covariance the Jacobean matrices are needed. G describes the advance from step to step and V matches the noise into the equation. This model has been simulated in Matlab (Figure 4.2 and was tested on the car (Figure 4.2 and Figure 4.2). Figure 4.2 shows the odometry message in RVIZ. Figure 4.2 shows the cars odometry output (red and blue) and the measured position of the car (green). It can be seen that the difference (in this case) between calculated and real position is a tenth of a meter . While testing it turns out that the steering angle of the car is bigger, when the car is driving backwards. So a calculation correction value (130%) is multiplied to the angle, to make calculations more accurate and keep the covariance lower. For implementing a fully working Kalman-filter the odometry data and the covariance must be changed. Normally in ROS this is done by using offset variables to save the distance between detected position and the predicted position. Anyway 3 new commands where included in the program to change the odometry data and the covariance of the car shown in Figure 4.2.

## 4.3 Control Interface

The terminal program is written in C++ with the boost library. Its base part is the serial interface for the communication with arduino. The interface consists of a serial reader in a single thread, which splits all messages at the command end delimeter which is '\n' and buffers all received commands in a fifo-buffer. Afterwords the main program processes the collected data in the loop. To write commands a non threaded serial writer is used. The terminal program reads ROS-messages and sends it to the Arduino program. For testing the car two different possibilities are prepared. To control the car by a joy-pad the node can directly read sensor-joy-messages. For controlling the car with another motion program it can read velocity messages from ROS. For this calculation the distance between the car's wheels is needed to calculate the steering angle depending to the angular velocity $asin(\frac{d_{wheels}*\omega}{v})$. Afterwards the terminal sends the command with the serial interface until the #vs and the #as response appears in the buffer. The program is also able to publish the IMU messages for the Razor IMU node like the Razor terminal program. Therefore a sensor_msgs::imu message is generated out of the captured #YPRAG command. Which consists of the angular-velocities, the linear-velocities, the orientation and it's covariances. To visualize this data messages simply the Razor IMU node is used. For publishing the odometry data a stamped odometry message is published on ROS containing x, y, z, a quaternion for the orientation

$$\Delta\theta = \frac{\delta t * v(t) * tan(\phi)}{d_{wheels}} \tag{4.1}$$

$$x_{odom} = x_{odom} + sin(\theta_{odom}) \tag{4.2}$$

$$x_{odom} = y_{odom} + cos(\theta_{odom}) \tag{4.3}$$

$$\theta_{odom} = \theta_{odom} + \Delta\theta \tag{4.4}$$

$$G = \begin{matrix} 1 & 0 & -\delta t * v(t) * sin(\theta_{odom}) \\ 0 & 1 & \delta t * v(t) * cos(\theta_{odom}) \\ 0 & 0 & 1 \end{matrix} \tag{4.5}$$

$$V = \begin{matrix} \Delta t * cos(\theta) & 0 \\ \Delta t * sin(\theta) & 0 \\ \frac{\Delta t * tan(\frac{\Delta\theta}{\Delta t})}{d_{wheels}} & \frac{\Delta t * v(t)}{cos(\frac{\Delta\theta}{\Delta t})^2} \end{matrix} \tag{4.6}$$

$$\Gamma = \begin{matrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{matrix} \tag{4.7}$$

$$\Sigma = G * \Sigma * G^T + V * \Gamma * V^T \tag{4.8}$$

Figure 4.3: Odometry calculation, Equation one to four are describing the position calculation and Equation five to eight are describing the motion covariance

and the car's velocities. Additionally every message is marked with the frame id for tf. This message describes the distance between the base_link of the system and the actual frame id which can be displayed for example with rviz.

## 4.4 Conclusion and Future Work

It has been shown how to build up a base frame for a autonomous race car. Therefore a brushless controller has been programmed and tested on an Arduino UNO and a Raspberry PI has been setted up with ROS to control the car. Furthermore an odometry model for an Ackerman drive has been derived and tested on the car. Additionally a IMU has been integrated in the car to get extra information about the cars position. This IMU can be used in further tasks to improve the accuracy of the motion model of the car. Therefore the calculated position from the odometry model can be merged with the determined position of the IMU. This should increase the positions accuracy and decrease the growth of the covariance of the position. Another point to improve is the base frame of the car. To decrease the steering 'noise' and get even more accurate localization values, the base frame can be improved mechanically. The brushless controller takes only care on the motors hall sensors. So the timing angle has to be guessed because no response from the current signal of the motor is given. To improve this controller additionally the

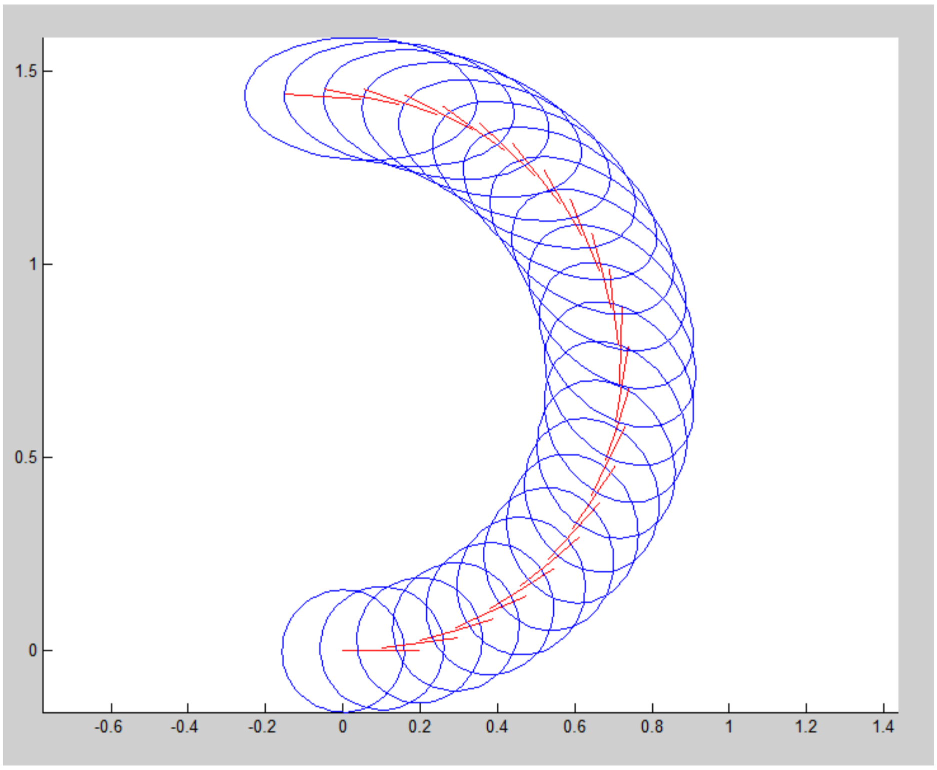
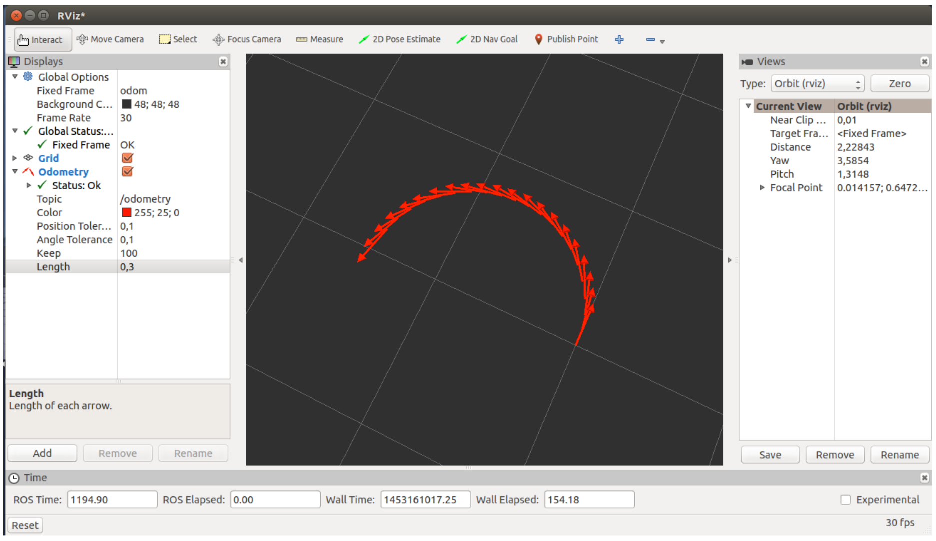


Figure 4.4: Odometry simulation

current signal can be monitored to improve the behaviour of the motor. In summary we can say the car and the localization are working well, but there are many more tasks to improve accuracy and behaviour of the car.



| | |
|---|---|
| Global Options | |
| Fixed Frame | odom |
| Background C… | ■ 48; 48; 48 |
| Frame Rate | 30 |
| ✓ Global Status:… | |
| ✓ Fixed Frame | OK |
| Grid | ✓ |
| Odometry | ✓ |
| ✓ Status: Ok | |
| Topic | /odometry |
| Color | 🟥 255; 25; 0 |
| Position Toler… | 0,1 |
| Angle Tolerance | 0,1 |
| Keep | 100 |
| Length | 0,3 |

**Length**
Length of each arrow.

Type: Orbit (rviz)

| | |
|---|---|
| Current View | Orbit (rviz) |
| Near Clip … | 0,01 |
| Target Fra… | <Fixed Frame> |
| Distance | 2,22843 |
| Yaw | 3,5854 |
| Pitch | 1,3148 |
| Focal Point | 0.014157; 0.6472… |

ROS Time: 1194.90 ROS Elapsed: 0.00 Wall Time: 1453161017.25 Wall Elapsed: 154.18

30 fps

Figure 4.5: Odometry test with RVIZ

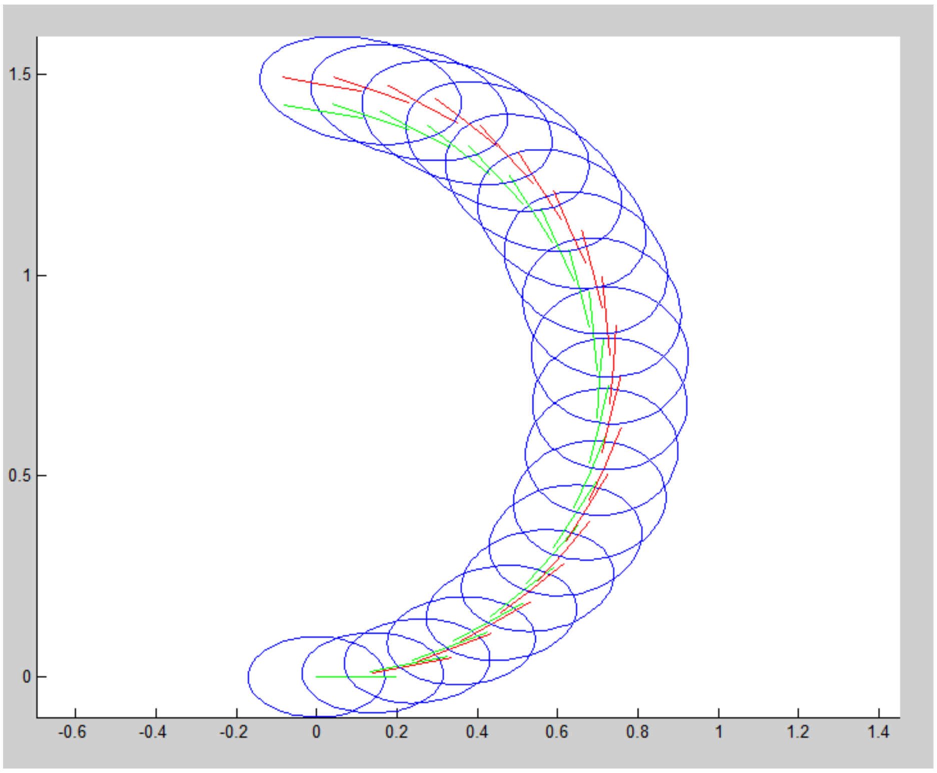


Figure 4.6: The Cars odometry data (red and blue) compared to the real position (green)

| command | arguments | function |
|---|---|---|
| #os | $x_{odom}$, $y_{odom}$, $\theta_{odom}$, $cov_{1,1}$, $cov_{1,2}$, $cov_{1,3}$ $cov_{2,1}$, $cov_{2,2}$, $cov_{2,3}$ $cov_{3,1}$, $cov_{3,2}$, $cov_{3,3}$ | sets odometry data and variance |
| #oso | $x_{odom}$, $y_{odom}$, $\theta_{odom}$, | sets odometry data |
| #osv | $cov_{1,1}$, $cov_{1,2}$, $cov_{1,3}$ $cov_{2,1}$, $cov_{2,2}$, $cov_{2,3}$ $cov_{3,1}$, $cov_{3,2}$, $cov_{3,3}$ | sets variance |

Figure 4.7: Odometry serial commands

# Bibliography

[AK14] T.Ramesh A. Kiruthika, R. Agasthiya. Speed control of a sensored brushless dc motor using flc. *International Journal of Engineering Research and Technology*, 2014.

[BÃij13] Roland BÃijchi. *Brushless-Motoren und -Regler.* Verlag fuer Technik und Handwerk neue Medien GmbH, 2013.

[Gam01] Salah Sukkarieh Gamini. The aiding of a low-cost strapdown inertial measurement unit using vehicle modele constraints for land vehicle application. *IEEE Transactions on robotics and automation*, 2001.

[Ng09] Morgan Quigley Brian Gerkey Ken Conley Josh Faust Tully Foote Jeremy Leibs Eric Berger Rob Wheeler Andrew Ng. Ros: an open-source robot operating system. *ICRA Workshop on Open Source Software*, 2009.

[SJ14] Madhurima Chattopadhyay Sharad Jaiswal, Debjyoti Chowdhury. Performance analysis of sensored and sensorless drive of bldc motor using different types of dc/dc converters in matlab/simulink platform. *International Journal of Electrical, Electronics and Data Communication*, 2014.

[VKSP13] A.K.Pandey Vinod KR Singh Patel. Modeling and performance analysis of pid controlled bldc motor and different schemes of pwm controlled bldc motor. *International Journal of Scientific and Research publications*, 2013.