

# HIGH SPEED NONHOLONOMIC MOBILE ROBOT ONLINE TRAJECTORY OPTIMATIZATION

by  
Ying Xu

*A thesis submitted to Johns Hopkins University in conformity with the  
requirements for the degree of Master of Science in Mechanical Engineering*

*Baltimore, Maryland*

October 2014

## *Abstract*

Autonomous Mobile robot navigation has become a popular topic in robotics due to its emerging applications in self-driving vehicles and autonomous air-drones. This essay explores the two main components of navigation namely perception and control. Simultaneous localization and mapping (SLAM) has been one of the most widely adopted collection of methods for self localization of robots in unknown environments. Here we put our focus on gMapping [1] [2] for offline map building and Adaptive Monte Carlo Localization (AMCL) [3] for realtime localization. On the control side, Forward Backward Sweep Method is used to generate locally optimal trajectories and the corresponding feedback control law. Our experiments show that using the above methods and a properly integrated system autonomous navigation can be achieved with up to 1 *m/s* navigation speed.

## *Acknowledgements*

I would like to express my gratitude to my advisor Professor Marin Kobilarov for his guidance and support on this project. I would also like to thank my fellow labmates Gowtham Sandilya and Subhransu Mishra for their generous help.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>Contents</b>	<b>iv</b>
<b>List of Figures</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 System Set-up	1
1.2 Mobile Base Hardware Set-up	2
1.2.1 Sensor Calibration	4
1.3 Software Set-up	5
1.4 Control and Safety Feature	6
<b>2 Vehicle Odometry</b>	<b>8</b>
<b>3 Localization And Mapping</b>	<b>15</b>
3.1 Map Building	15
3.2 AMCL	19
<b>4 Trajectory Planning And Tracking</b>	<b>23</b>
4.1 Dynamic Feedback Linearization	23
4.1.1 Feedback Linearization for Simple Car Model	23
4.2 Forward-Backward Sweep Method	25
<b>5 Results And Discussion</b>	<b>28</b>
5.1 Experiment Description and Results	28
5.2 Future Work	29
<b>6 Conclusion</b>	<b>31</b>
<b>A Experiment procedure</b>	<b>32</b>
<b>B System Frames</b>	<b>33</b>

*Contents*

<b>C Forward Backward Sweep Method</b>	<b>35</b>
<b>Bibliography</b>	<b>37</b>
<b>Biographical Statement</b>	<b>38</b>

# List of Figures

1.1	RC car test platform . . . . .	2
1.2	System Flowchart . . . . .	3
1.3	Input calibration . . . . .	4
1.4	Motion Capture Distance . . . . .	4
1.5	Steering Calibration . . . . .	5
1.6	User Interface . . . . .	7
2.1	simple car model . . . . .	9
2.2	Odometry Calibration Result Without Gyro (X) . . . . .	10
2.3	Odometry Calibration Result Without Gyro (Y) . . . . .	10
2.4	Odometry Calibration Result Without Gyro ( $\theta$ ) . . . . .	11
2.5	Odometry Calibration Result With Gyro (X) . . . . .	11
2.6	Odometry Calibration Result With Gyro (Y) . . . . .	12
2.7	Odometry Calibration Result With Gyro ( $\theta$ ) . . . . .	12
2.8	Odometry Calibration Result With Curve Length . . . . .	13
2.9	Odometry Calibration Result With With Curve Length (Y) . . . . .	14
2.10	Odometry Calibration Result With With Curve Length( $\theta$ ) . . . . .	14
3.1	Kinect Framework . . . . .	18
3.2	Generate Occupancy Grid Map of Experimental Hallway . . . . .	19
3.3	AMCL In Motion . . . . .	20
3.4	AMCL Localization . . . . .	21
3.5	Actual Robot location . . . . .	21
3.6	AMCL Odometry . . . . .	22
4.1	Feedback Linearization controller simulation result . . . . .	25
4.2	Feedback Linearization controller simulation result . . . . .	27
5.1	Computed vs. Actual Trajectory in Controller Testing . . . . .	28
5.2	Video Screenshot . . . . .	29
5.3	issue with trajectory . . . . .	30
B.1	Position of The Vehicle Relative to The World . . . . .	33
B.2	Vehicle Body Frames . . . . .	34
B.3	system frames . . . . .	34

# Chapter 1

## Introduction

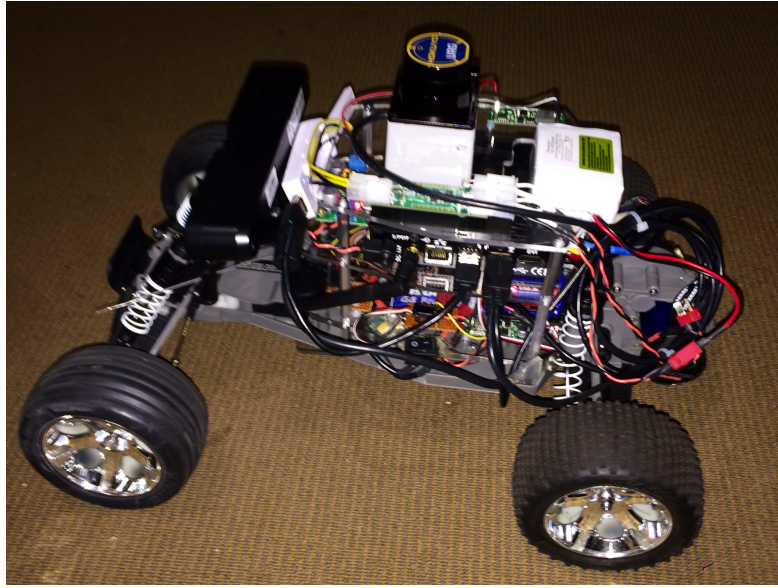
With the rapid development of sensor technologies and onboard computing power, autonomous navigation has become an active area of research. From Google's self-driving car to the iRobot home cleaning assistants, emerging applications utilizing technologies from this field are coming into people's everyday lives. Being two main consisting parts of autonomous navigation, environmental perception and automatic control has been studied extensively over the past few years and have resulted in some highly applicable methods. The goal of this essay is to describe our efforts in integration of a mobile platform in hardware and software that utilizes carefully chosen and implemented perception and control techniques to achieve fully autonomous navigation at a relatively high speed and low cost. The remainder of this chapter introduces in detail of our high level system setup.

### 1.1 System Set-up

The first step is to set up an experimental platform that allows for algorithm testing. Two major test environment are evaluated here with the first that uses motion capture for localization in an indoor environment and the second in the hallway that uses SLAM (Simultaneous Localization and Mapping) for localization.

The purchased RC car (named Rustler shown in figure 1 above) contains mechanical frames and built-in motors and controller box, which is much easier to work with than a custom designed mobile system, but the built-in controller box has limited speed control accessibility. This mobile car is a non-holonomic system that makes it possible to test rear-wheel driven vehicle control algorithms on a much affordable platform. With suspension on Rustler, this system has the ability to run in outdoor terrains which extends this platform to testing of control and optimization methods from 2D to 3D.

FIGURE 1.1: RC car test platform



The following sections describes this experimental platform in terms of hardware and software set-up.

## 1.2 Mobile Base Hardware Set-up

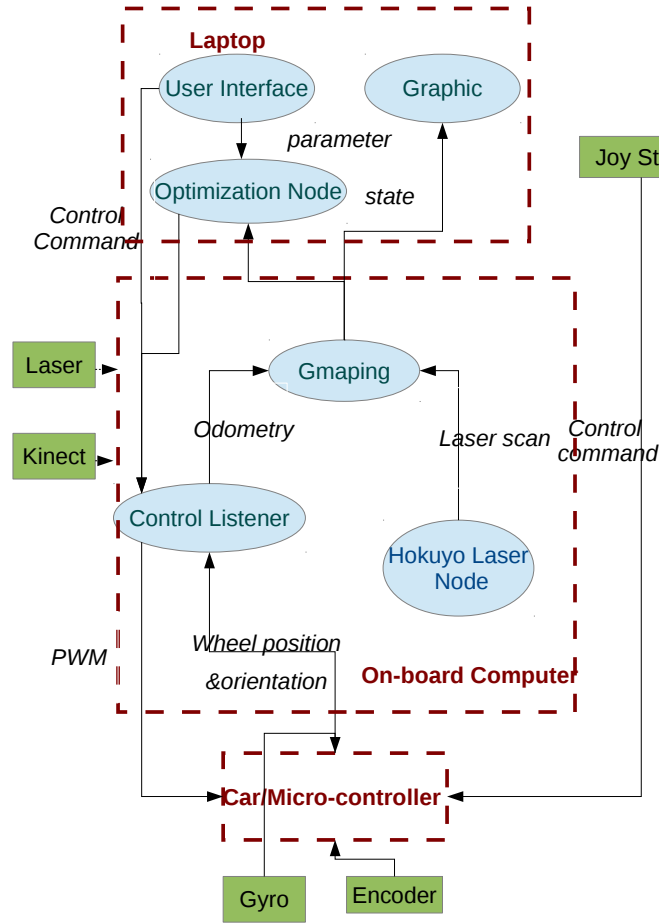
The experimental mobile platform contains a Traxxas Nitro Rustler car base with brushless back driving motor, and front servo motor for steering. The system runs on APC2 computer with Intel® Atom™ Processor Z550 (512K Cache, 2.00 GHz, 533 MHz FSB-) and DELL Vostro laptop with Intel(R) Core(TM) i7-3612QM CPU @ 2.10GHz.

Onboard sensors and actuators include:

- URG-04LX scanning laser range finder with 240 deg measuring area and 60 to 10000mm measurement range.
- ASUS Xtion PRO live is the vision sensor that used for slam or video stream.
- ATM203 Encoder is high accuracy axial modular encoder, which has a build in SPI communication board. The encoder is mounted on the back wheel records the absolute movement of the back wheel.
- 5GHz Wifi adaptor is used to separate wifi signal. Communication between on board computer and laptop is based on wifi connection. Currently, most wifi signal is 2.4GHz, so this system used 5GHz wifi that can avoid interference from most of the wif signals.



FIGURE 1.2: System Flowchart



- Li-Ion Polymer 7.4V 3 cell battery for computer power supply.
- 3900mAh 3 cell 11.1V G8 Pro battery for electronics power supply.
- Pololu 5V Stepdown Volt Reg converts the voltage output from battery to 5V input voltage to micro-controller.
- Y-PWR, Hot Swap, Load Sharing Controller is ideal diode that two power sources can be used together. This allows user to switch the power source without shutting down the system.
- Arduino USB to Serial converter is used to connect micro-controller to onboard controller. But power pin wasn't connect to computer. Electronics and computer's power supply were separated to prevent restart of micro controller while the ROS node start on computer or unstable random signal from power pin that leads to restart of the micro controller board.

- BeStar Acoustic Component (F-B-P3009EPB-01 LF) is sound actuator that beeps when the battery runs low either on computer or electronics.
- Futaba R617FS 7-Channel 2.4GHz FASST Receiver receives control command from Joystick.
- FUtaba T7C Joystick allows user manual remote control.

### 1.2.1 Sensor Calibration

In order for subsequent algorithm testings to accurately reflect their performance, careful calibration of the onboard sensors is key. One of the first steps is to find a conversion between the controller PWM signal to car speed and Steering angle. Motion capture can measure the mobile robot position  $(x, y, \theta)$  at 125 *hz* and is used here as the ground truth. PWM signals from 1500 to 1800 with increments of 10 are fed as input commands to the driving motor, while the steering servo motor remains neutral. Rustler will drive at constant speed for around 1.5 m. Then, for each PWM input signal driving distance can be estimated from motion capture  $D = (x^2 + y^2)^{\frac{1}{2}}$ , and the corresponding driving speed can be estimated from the slope.

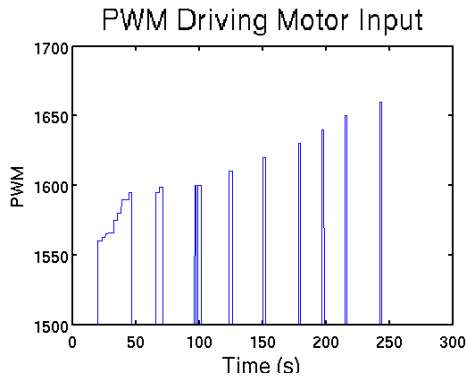


FIGURE 1.3: Input calibration

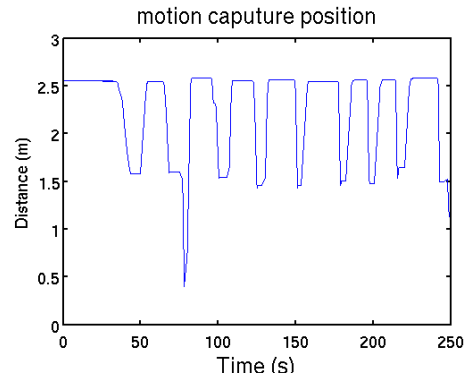


FIGURE 1.4: Motion Capture Distance

Estimation data at vehicle startup are ignored given that it takes time for the motor to accelerate to certain speed. As can be seen from 1.3 and 1.4, each input PWM is correlated to one spike. The conversion factor can be obtain from curve fitting the measured position data. Steering angle calibration utilizes very similar method (see figure 1.5 for a simplified model of the vehicle). Commands are sent to the steering motor with constant driving speed. The car will then drive in a circle and motion capture will again log its position information. With know rear distance  $L$ , and driving circle radius  $(\delta)$ , steering is calculated as equation 1.1. Due to the fact that both steering wheels are controlled by the same servo,  $\delta$  is consider same for both wheel.



package that takes in odometry data and laser scan and outputs estimated robot location related to map.

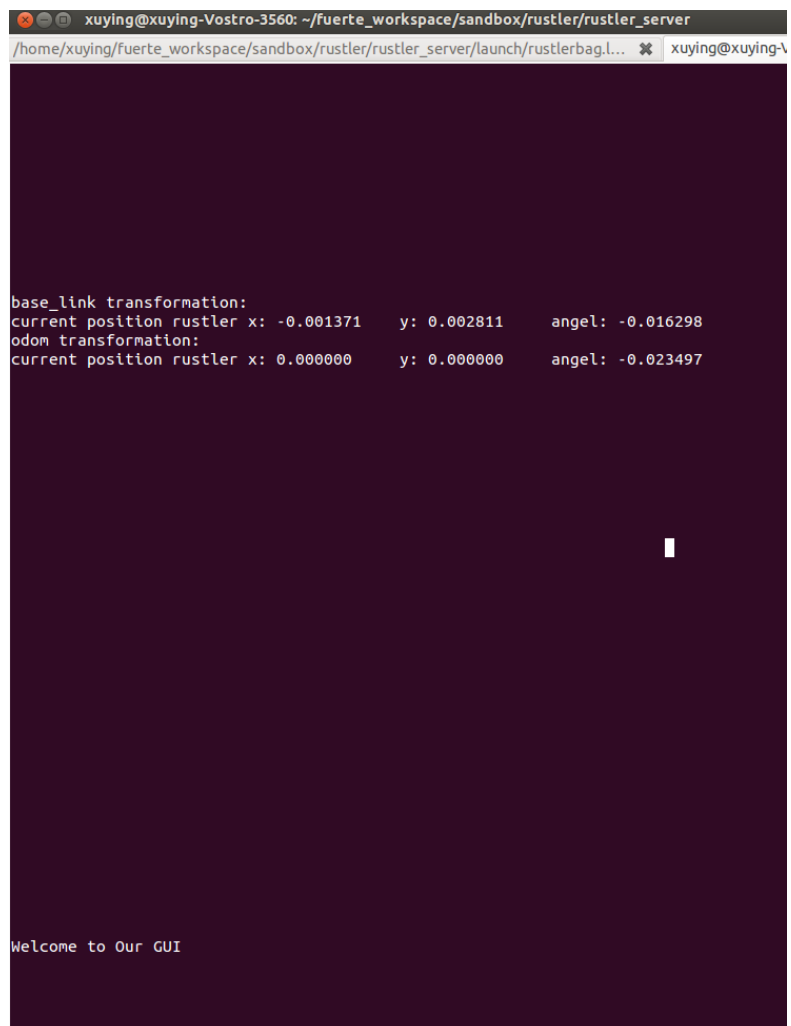
Three nodes run on on-board computer: Gcop, Setwaypoint and Rviz. Gcop uses General Linear-Quadratic Regulator to compute local optimal path (discussed more in later chapters). Setwaypoint takes in pre-designed global path and publish way points based on robot current location. Rviz takes in whole frames and generate 3D graphic display including the global map, robot frames and local optimal path.

## 1.4 Control and Safety Feature

Since the vehicle is a high powered test platform that runs at relatively high speed, making sure that the system can shut-down appropriately is a essential before starting any automatic navigation testing.

1. The first part is joystick control. Normally, the Futaba T7C joystick is used for AR.Drone control since it has a stable long range of round 50 meters. Standard APM\_RC\_APM2 library is used here for radio signal receiving. The microcontroller will only activate the motor and servo when it receives radio signal from the joystick. Radio has 7 channels, 2 switch has to be turned on together to toggle the vehicle to automatic driving mode, the rest of the channels are used for manual joystick driving. The Microcontroller will sent continuous neutral PWM signals (1500) to motor when the radio is turned on but not giving control signals to make sure the motor won't run under random signal.
2. NCURSES terminal UI (figure 1.6) is a simple user interface that displays the current states  $(x, y, \theta)$  of the vehicle. All ROS nodes will be shut-down and the motor will detached from the micro controller upon termination (currently set to key "Q"). "D" is for command desired trajectory and "I" for user input continuous PWM signals.
3. Battery switch is connected directly from battery to electronics. Since previous controls is either based on radio or wifi communication, a hard stop is designed here to shut down the system power in case any communication issues occurs. This switch will only shutdown the power supply to the electronics and micro-controller, thus the onboard computer will remain running. However, no power line were connect between the host computer and onboard computer, thus restarting the electronics won't affect ROS nodes on the host computer.

FIGURE 1.6: User Interface



A terminal window with a dark purple background and white text. The window title bar shows the user 'xuying' on a machine named 'xuying-Vostro-3560' in the directory '~/fuerte\_workspace/sandbox/rustler/rustler\_server'. The terminal output displays transformation data for 'base\_link' and 'odom', including 'current position rustler' coordinates (x, y) and 'angel' values. At the bottom, it says 'Welcome to Our GUI'.

```
xuying@xuying-Vostro-3560: ~/fuerte_workspace/sandbox/rustler/rustler_server
/home/xuying/fuerte_workspace/sandbox/rustler/rustler_server/launch/rustlerbag.l... xuying@xuying-
base_link transformation:
current position rustler x: -0.001371    y: 0.002811    angel: -0.016298
odom transformation:
current position rustler x: 0.000000    y: 0.000000    angel: -0.023497

Welcome to Our GUI
```

## Chapter 2

# Vehicle Odometry

Odometry is the use of motion sensor information for estimation of position over time. On our mobile platform, odometry is obtained from a combination of encoder and gyro sensor data. Here the position of the back wheels can be obtained from encoders and vehicle's orientation can be obtained by processing gyro data. Although odometry will diverge due to sensor drift, this divergence can be compensated by a measurement update (details in the next chapter). But getting an accurate odometry estimate is an essential first step for localization.

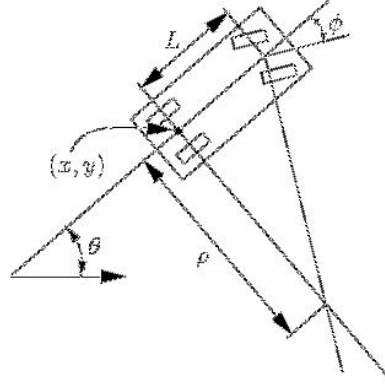
The vehicle's states are usually presented as

$$\begin{pmatrix} x \\ y \\ \theta \\ v \\ \omega \end{pmatrix} \quad (2.1)$$

where  $x$  and  $y$  are vehicle's planar location and  $\theta$  is the heading all in world coordinate.  $v$  and  $\omega$  represents linear and rotational velocity. The pair of control inputs is the driving velocity and steering angle

$$\begin{pmatrix} u \\ \phi \end{pmatrix} \quad (2.2)$$

FIGURE 2.1: simple car model



The simple car model (illustrated in figure 2.1 ) can be written as

$$\begin{pmatrix} \dot{x} = u * \cos(\theta) \\ \dot{y} = u * \sin(\theta) \\ \dot{\theta} = \frac{u}{L} \tan(\phi) \end{pmatrix} \quad (2.3)$$

Assuming that sampling can be done at a fast rate, the position of the vehicle can be reasonably estimated by

$$\begin{pmatrix} x = x + \dot{x} \delta t \\ y = y + \dot{y} \delta t \\ \theta = \theta + \dot{\theta} * \delta t \end{pmatrix} \quad (2.4)$$

$\phi$  in equation 2.3 is the vehicle steering angle which can be obtain from the gyro.  $D_{is}$  are the discrete encoder readings. In our first attempt to acquire accurate odometry data, wheel speed  $u$  is obtained by finite differencing encoder readings, while front servo PWM signals give turning angle  $\phi$ . The dynamics update equations are given below

$$\begin{aligned} u &= (D' - D) / \delta t \\ \theta' &= \theta + \frac{u}{L} \tan \phi \delta t \\ x' &= x + u \cos(\theta) \delta t \\ y' &= y + u \sin(\theta) \delta t \end{aligned} \quad (2.5)$$

where  $[x', y', \theta']$  represents the states at the next time step. A few issues arise from this treatment. This model doesn't take into account the fact that the steering angle command

FIGURE 2.2: Odometry Calibration Result Without Gyro (X)

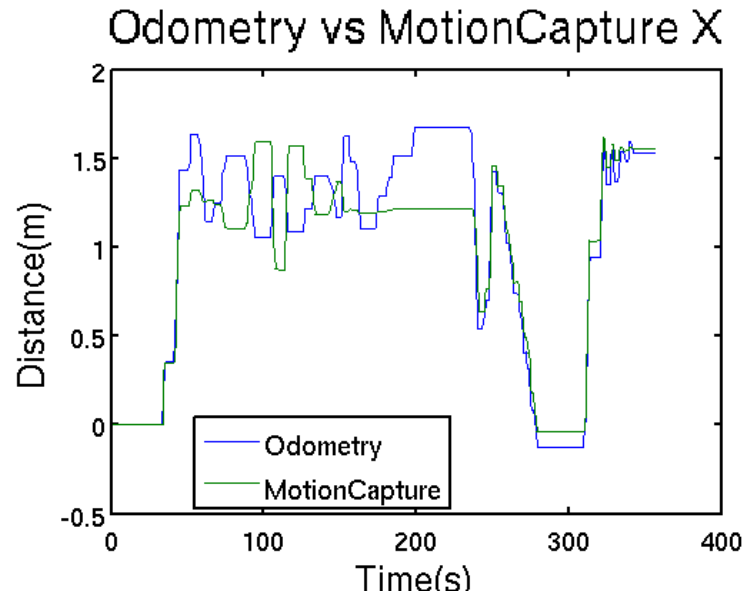
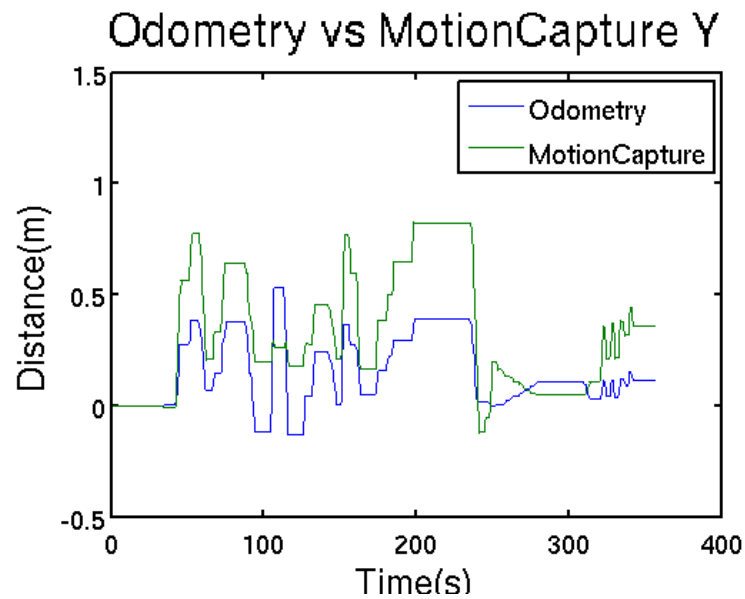


FIGURE 2.3: Odometry Calibration Result Without Gyro (Y)



signals does not transfer flawlessly to the actual steering angle due to a number of factor including ground and mechanism friction, signal noise as well as inertial effects. Figures 2.2, 2.3, 2.4 below illustrates the odometry data obtained using the above formulation compared to the ground truth obtained from motion capture. We can see that although the data trend is correct most of the time (less so for X), there exists large error.

Our next attempt integrates the gyro readings into the system and  $\theta$  is obtained directly by numerically integrating the gyro measurement  $\omega$ . Results are plotted in figures 2.5, 2.6, 2.7 which shows a much more promising tracking performance.



FIGURE 2.4: Odometry Calibration Result Without Gyro ( $\theta$ )

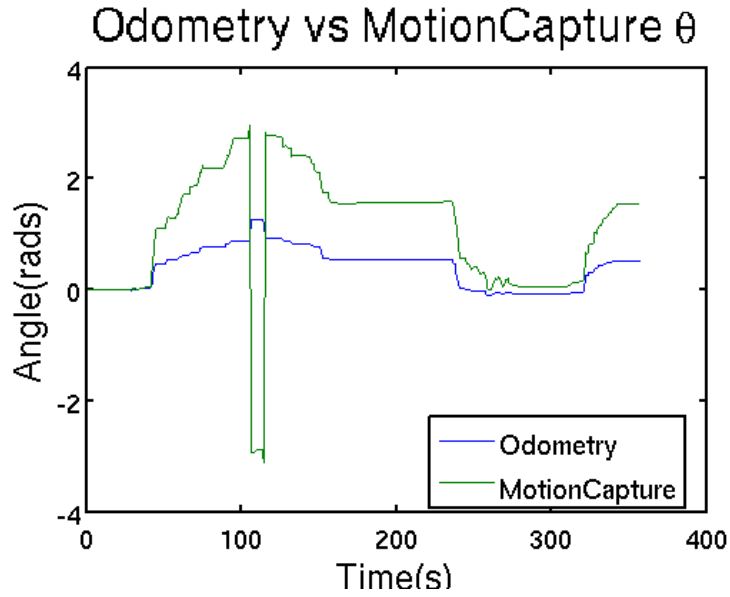
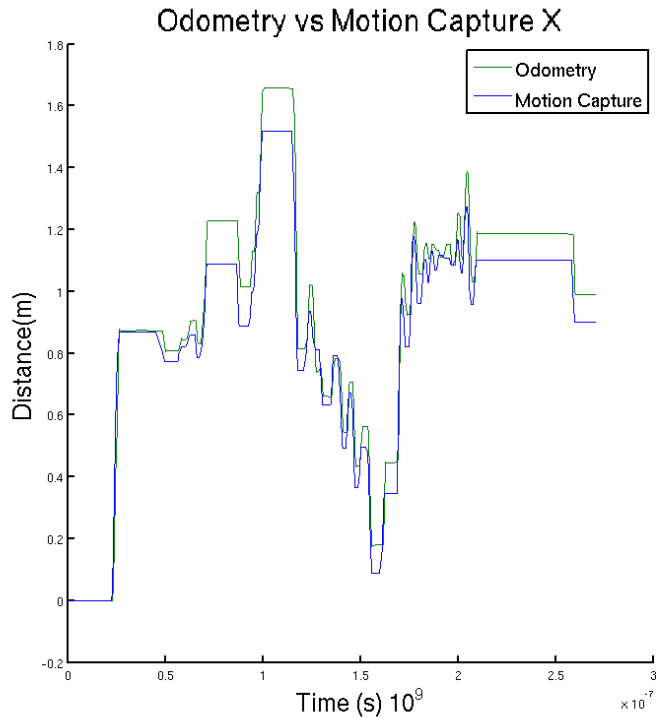


FIGURE 2.5: Odometry Calibration Result With Gyro (X)



In the last attempt, we aimed for a smoother interpolation between gyro and encoder readings. Therefore 10 integration steps are used between successive sensor readings. Assume that for the small time interval  $t_0$  to  $t_1$  the gyro measurement stayed constant at  $\omega$ . It is desirable to find an expression that can estimate the vehicle location at time  $t \in [t_0, t_1]$ . Looking at the expression for  $x$  we have

FIGURE 2.6: Odometry Calibration Result With Gyro (Y)

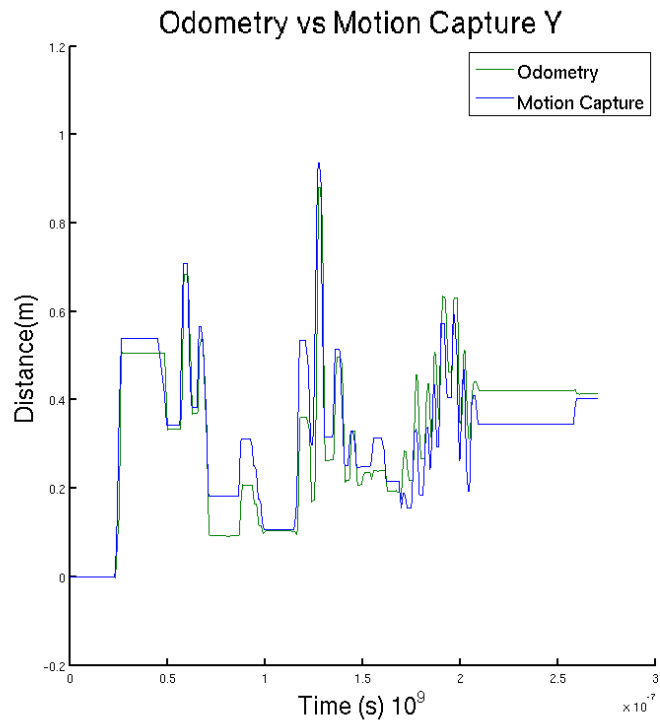


FIGURE 2.7: Odometry Calibration Result With Gyro ( $\theta$ )

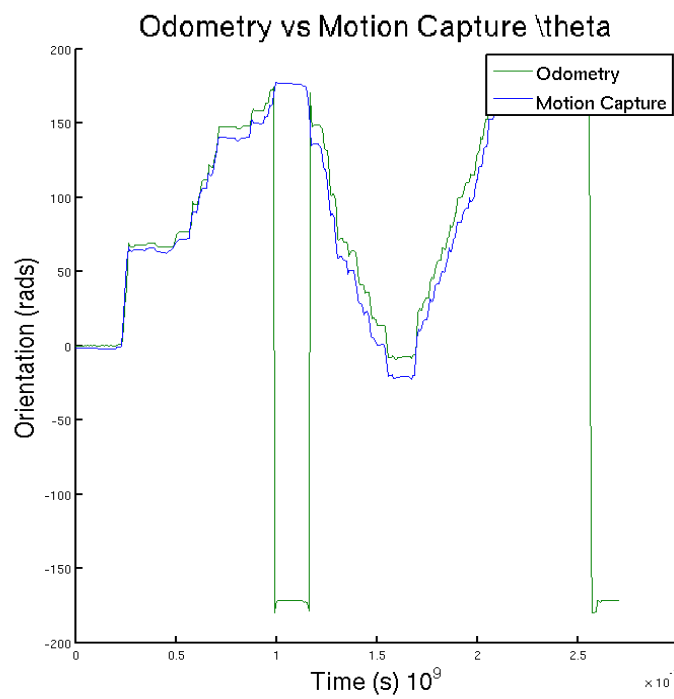
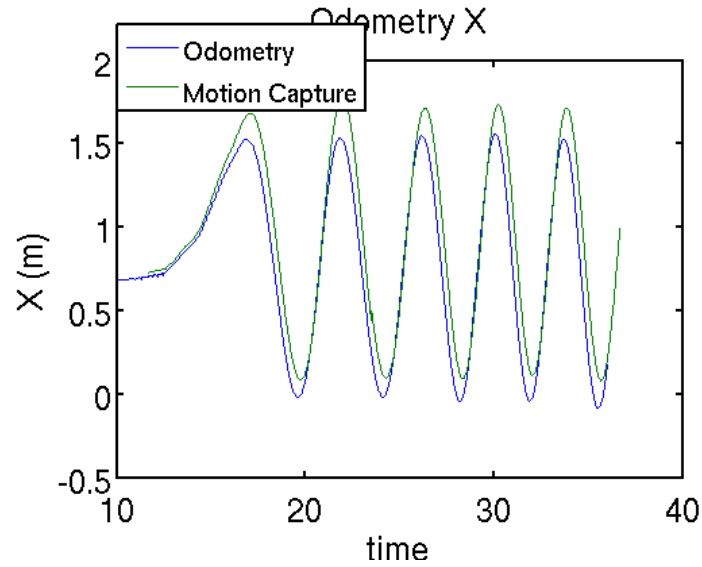


FIGURE 2.8: Odometry Calibration Result With Curve Length



$$\begin{aligned}
 x(t) &= x(t_0) + \int_{t_0}^{t_1} u \cos(\theta(t)) d\tau \\
 &= x(t_0) + \int_{t_0}^{t_1} u \cos(\theta(t_0) + \omega(\tau - t_0)) d\tau \\
 &= x(t_0) + \frac{u}{\omega} (\sin(\theta(t_0) + \omega(t - t_0)) - \sin(\theta(t_0)))
 \end{aligned} \tag{2.6}$$

the same can be done for  $y$  by

$$y(t) = y(t_0) + \frac{u}{\omega} (-\cos(\theta(t_0) + \omega(t - t_0)) + \cos(\theta(t_0))) \tag{2.7}$$

Given such, as long as the sensor inputs stays constant, position estimates can be obtained for any time and can therefore be used to smooth out the trajectory between sensor measurements. A problem for this method is gyro drifting. After 10 mins running, gyro error become significant. One feature is implemented as a temporary solution where the user can use the provided UI or joystick to stop the vehicle for 5 sec, read in sensor reading and reinitialize all the state.

FIGURE 2.9: Odometry Calibration Result With With Curve Length (Y)

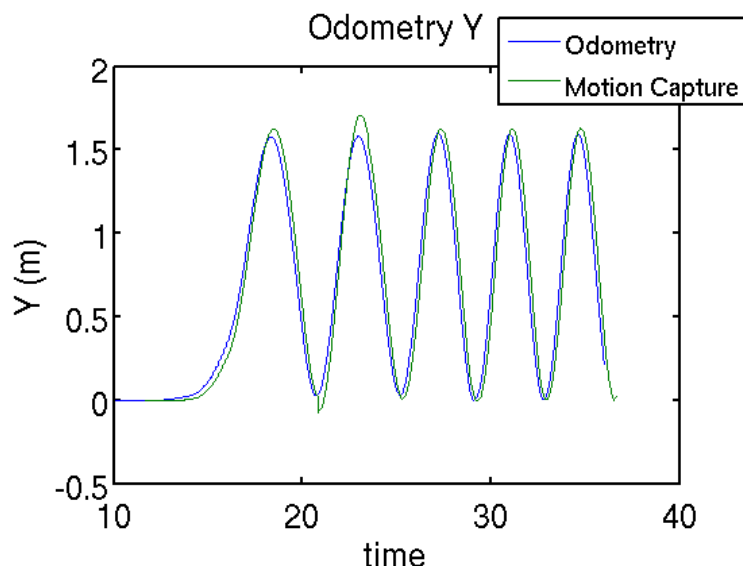
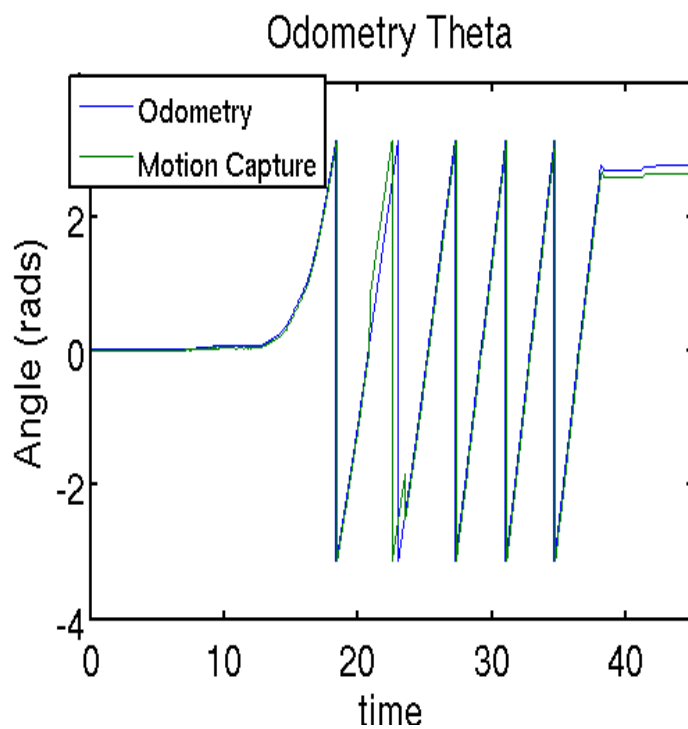


FIGURE 2.10: Odometry Calibration Result With With Curve Length( $\theta$ )



## Chapter 3

# Localization And Mapping

Mapping building is the first step of any fully autonomous driving operations. The vehicle needs to understand the environment to be able to navigate. Simultaneous Localization and Mapping also known as SLAM is widely used for such map building applications.

### 3.1 Map Building

In this project, the ROS package *slam\_gmapping* is used to create an occupancy grid map. The algorithm utilizes an improved version of the Rao-Blackwellized Particle Filter for learning grid maps that results in the reduction of sampling size, thus effectively reducing computation time.

For the general problem of mapping with particle filters, the goal is to estimate the joint posterior  $p(x_{1:t}, m | z_{1:t}, u_{1:t-1})$  for the map  $m$  and trajectory  $x_{1:t}$ . The Rao-Blackwellized particle filter simplifies the problem by making the factorization

$$p(x_{1:t}, m | z_{1:t}, u_{1:t-1}) = p(m | x_{1:t}, z_{1:t}) \cdot p(x_{1:t} | z_{1:t}, u_{1:t-1}) \quad (3.1)$$

This factorization allows us to decouple the process of estimating the robot's trajectory and the map which greatly simplifies the problem. Given the above, the SLAM problem can be categorized into a four-step process of *sampling - importance weighting - resampling - map estimation*. Because the length of trajectory increases over time, the importance weighting step given by

$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)} | z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)} | z_{1:t}, y_{1:t-1})} \quad (3.2)$$

can be very inefficient ( $\pi$  is the proposal distribution and  $p$  is the target distribution). However if the proposal distribution can be restricted to fulfill

$$\pi(x_{1:t} | z_{1:t}, u_{1:t-1}) = \pi(x_t | x_{1:t-1}, z_{1:t-1}, u_{1:t-1}) \cdot \pi(x_{1:t-1} | z_{1:t-1}, u_{1:t-2}) \quad (3.3)$$

then the weighting process can be massaged to a recursive form given by

$$\begin{aligned} w_t^i &= \frac{\eta p(z_t | x_{1:t}^i, z_{1:t-1}) p(x_t^i | x_{t-1}^i, u_{t-1})}{\pi(x_t^i | x_{1:t-1}^i, z_{1:t-1}, u_{1:t-1})} \cdot \frac{p(x_{1:t-1}^i | z_{1:t-1}, y_{1:t-2})}{\pi(x_{1:t-1}^i | z_{1:t-1}, u_{1:t-2})} \\ &\propto \frac{p(z_t | m_{t-1}^i, x_t^i) p(x_t^i | x_{t-1}^i, u_{t-1})}{\pi(x_t | x_{1:t-1}^i, z_{1:t-1}, u_{1:t-1})} \cdot w_{t-1}^i \end{aligned} \quad (3.4)$$

The contribution of *gmaaping* over conventional sampling based SLAM techniques with grid maps is twofold. First, the proposal distribution is modified to consider the most recent sensor measurement (if accurate) and sample only around that range. Second is the introduction of an adaptive resampling technique based on the effective sample size ( $N_{eff}$ ) measure to maintain a reasonable variety of samples. The proposal of the first idea (modified proposal distribution) is motivated by the fact that if the measurement model is highly concentrated (very precise) whereas the the proposal distribution is highly distributed, it would take an enormous number of particles to produce a set of weights that can accurately capture the target distribution. Assuming the target distribution is unimodal, define an interval  $L^i$  by

$$L^i = \{x | p(z_t |_{t-1}^i, x) > \epsilon\} \quad (3.5)$$

and locally approximate the proposal distribution  $p(x_t | m_{t-1}^i, x_{t-1}^i, z_t, u_{t-1})$  around the maximum of the likelihood function as a Gaussian. Then for each particle, a scan-matching algorithm can be used to provide the pose of maximum likelihood given the map  $m_{t-1}^i$  and an initial guess, and in an interval (given by equation 3.5) around that point a set of sampling points can be selected and their mean and covariance can be calculated by

$$u_t^i = \frac{1}{\eta^i} = \sum_{j=1}^K x_j \cdot p(z_t | m_{t-1}^i, x_j) \cdot p(x_j | x_{t-1}^i, y_{t-1}) \quad (3.6)$$

$$\begin{aligned} \Sigma_t^i = \frac{1}{\eta^i} \cdot \sum_{j=1}^K p(z_t | m_{t-1}^i, x_j) \cdot p(x_j | x_{t-1}^i, u_{t-1}) \\ \cdot (x_j - \mu_t^i)(x_j - \mu_t^i)^T \end{aligned} \quad (3.7)$$

where

$$\eta^i = \sum_{j=1}^K p(z_t | m_{t-1}^i, x_j) \cdot p(x_j | x_{t-1}^i, u_{t-1}) \quad (3.8)$$

and the new pose of particle  $i$  can be drawn from this improved proposal distribution (which also proves to be the optimal proposal distribution). And using newly generated particles the weights can be computed as

$$\begin{aligned} w_t^i &= w_{t-1}^i \cdot p(z_t | m_{t-1}^i, x_{t-1}^i, u_{t-1}^i) \\ &= w_{t-1}^i \cdot \int p(z_t | m_{t-1}^i, x') \cdot p(x' | x_{t-1}^i, u_{t-1}^i) dx \\ &\simeq w_{t-1}^i \cdot \sum_{j=1}^K p(z_t | m_{t-1}^i, x_j) \cdot p(x_j | x_{t-1}^i, u_{t-1}^i) = w_{t-1}^i \cdot \eta^i \end{aligned} \quad (3.9)$$

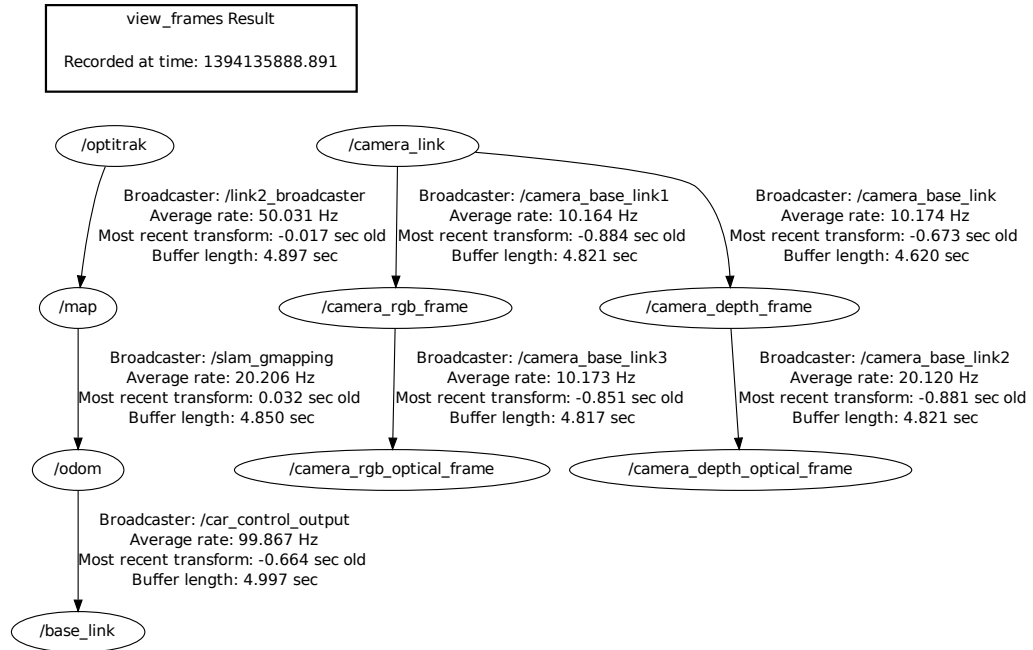
For the adaptive resampling process, the effective sample size given by

$$N_{eff} = \frac{1}{\sum_{i=1}^N (\tilde{w}^i)^2} \quad (3.10)$$

is used to evaluate the performance of the current sample set in terms of representing the target posterior.  $N_{eff}$  can be viewed as a measure of dispersion of the weights (similar to covariance), if  $N_{eff}$  is too large, that means the estimated target distribution is overly concentrated (or over-confident) which will result in the newly sampled particles to lose diversity and therefore could lead to localization failures. Therefore resampling is only executed with  $N_{eff}$  is below a threshold thereby not only preserving the variety of particles but also reduces computational cost.

However, even with *gmapping*, the speed of realtime map generation is limited (map update speed is 1 Hz). Therefore, for the purpose of testing our control algorithm, a map of the experimental environment is built in advance using the *slam\_gmapping* package

FIGURE 3.1: Kinect Framework



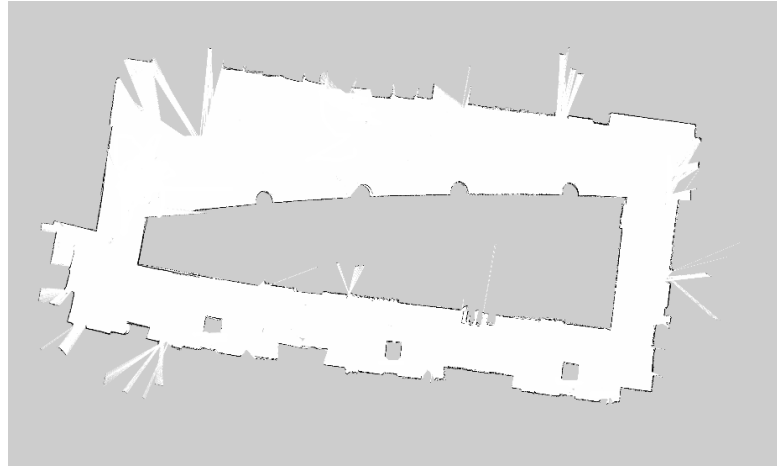
and the AMCL(Adaptive\_MCL) package is then used for perception and provide the vehicle's position state estimates to the controller.

Two sensor are tested here for localization. The first is Kinect which is a low cost motion sensor very popular for localization and mapping operations. The OpenNi ROS node is well developed for the Kinect sensor with built-in sensor driver and image streaming tools (package contained ROS nodes shown in figure 3.1). Here the Kinect point cloud depth information is used to simulate laser information which makes implementation of 2D SLAM methods possible. In addition, Kinect also has stereo vision and color camera, that gives it the potential to obtain 3D map using RGBDSlam. Low sensor detection range processing speed (10 fps) and resolution poses great limitation on the usage of this sensor. Due to the above, a map(shown in figure 3.2) is generated by using the *slam\_gmapping* package offline with the odometry and scan data logged from teleoperating the vehicle.

The second is a Hokuyo laser range finder that gives a deg240 detection angle range and 10 *hz* of measurement frequency. This performance is well suited for the online localization task that serves as the perception aspect of the controller



FIGURE 3.2: Generate Occupancy Grid Map of Experimental Hallway



## 3.2 AMCL

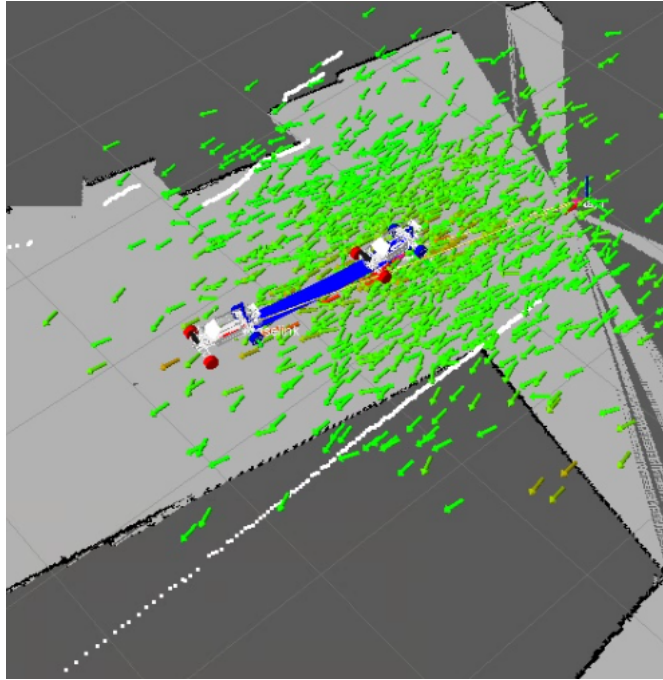
The Adaptive Monte Carlo Localization (AMCL) is a localization package that implements the KLD-Sampling MCL Method for position tracking in 2D. The key to this approach is to use a particle filter to adaptively vary the size of the sample set. The result is a small sample size if the target distribution is tightly focused around a small region and a larger sample size if it is distributed over a larger area (higher uncertainty), and therefore bound the approximation error. Such an error is measured by the Kullback-Leibler distance that measures the difference between two probability distributions  $p$  and  $q$  given by

$$K(p, q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \quad (3.11)$$

and hence the name KLD. Although the KL-distance is not a metric, it is widely adopted as a standard measure for evaluating such differences. Refer to [4] for more details on the approach. In addition, the AMCL package utilizes assistive algorithms from [3] such as *sample\_motion\_model\_odometry*, *beam\_range\_finder\_model*, *likelihood\_field\_range\_finder\_model*, *Augmented\_MCL*, and *KLD\_Sampling\_MCL*. The open source implementation of AMCL is published as a ROS package that takes in the laser scan message, necessary transforms, an initial position and an occupancy grid map, and the outputs will be the estimated robot pose, the estimated pose information of a set of particles maintained by the algorithm as well as the transforms necessary to map the results.

Error model is key to insure localization. Here, error model were obtained experimentally by compare the Odometry result with motion capture. Base on sensor rate 10 *hz*, error model is compute as below:

FIGURE 3.3: AMCL In Motion



alpha1: Specifies the expected noise in odometry's rotation estimate from the rotational component of the robot's motion.  $\frac{\Delta\theta_{Odom} - \Delta\theta_{motionCapture}}{\Delta\theta_{motionCapture}} = 0.08$

alpha2: Specifies the expected noise in odometry's rotation estimate from translational component of the robot's motion.  $\frac{\Delta\theta_{Odom} - \Delta\theta_{motionCapture}}{\Delta dis_{motionCapture}} = 0.2$

alpha3: Specifies the expected noise in odometry's translation estimate from the translational component of the robot's motion.  $\frac{\Delta dis_{Odom} - \Delta dis_{motionCapture}}{\Delta dis_{motionCapture}} = 0.8$

alpha4: Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion.  $\frac{\Delta dis_{Odom} - \Delta dis_{motionCapture}}{\Delta dis_{motionCapture}} = 0.4$

Because the experiment takes place in the hallway where no motion capture or other method can provide exact location information. Figure 3.3 provides a screenshot of the system in operation. The vehicle in the back represents the current position (as estimated by amcl) while the vehicle in front is the goal position. The white lines are laser readings of the environment. And the green arrows are the particles performing the localization task. The particles eventually converge to a small region. Figure below shows the location and orientation result from AMCL method and compares it with a picture of the physical surroundings. In figure 3.4, the red frame is the world frame, blue is the vehicle frame, white areas represent unoccupied space whereas black and grey areas are occupied by obstacles. Comparing it with figure 3.5 we can observe that the the generated map can reasonably represent the physical world.

---

Although various localization algorithms are available, due to the system requirements

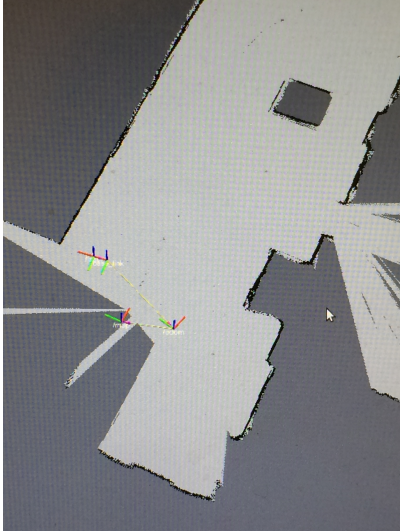


FIGURE 3.4: AMCL Localization

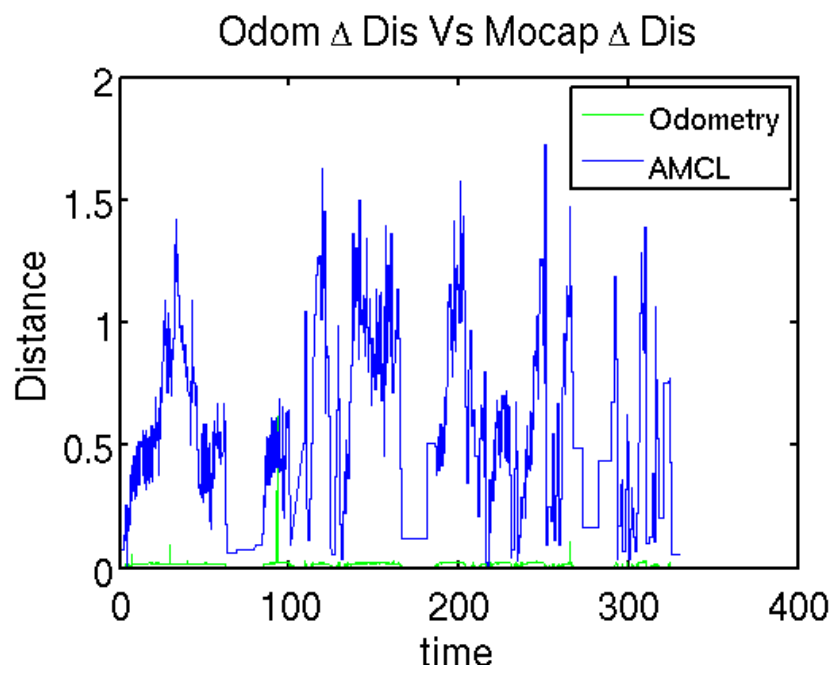


FIGURE 3.5: Actual Robot location

and limitations, we found AMCL to be the most applicable package for this application. Vehicle will mostly navigate in a planar region, thus a 2D map will be able to handle the localization task. However, since AMCL doesn't update the map, it is assumed for now that navigation is done in a static environment. An implementation of the RGDB SLAM is tested using Kinect but the image acquisition speed is limited to 2 *hz* which is not enough for high speed online trajectory optimization, whereas the AMCL update speed is based on laser data that is able to reach a 10 *hz* acquisition speed. Therefore based on feasibility and system performance we have decided to use AMCL as our localization method.

The major speed limitation is due to localization issue with high speed. Base on the experiment data, Odometry accuracy doesn't related to vehicle speed. However, the error model might be the cause of inaccurate localization. Experiment were tested to validate this assumption. In amcl, error model were setted as 500%,3.6 that the actual error is larger than the experimental obtained error model.

FIGURE 3.6: AMCL Odometry



## Chapter 4

# Trajectory Planning And Tracking

This chapter serves as the theoretical core of this essay. In subsequent sections the underlying methods for optimal trajectory planning and tracking of the generated trajectory are introduced along with a simulation of the optimal solver.

### 4.1 Dynamic Feedback Linearization

Dynamic Feedback linearization is a trajectory tracking approach that algebraically transforms non-linear systems into linear systems, which proceeds by differentiating the output  $y = h(x)$  enough time that the input appears linear and non-singular to the output. By then the transformed linear system can be controlled using classical methods. This is a easy-to-implement trajectory tracking scheme is used for various validation tests at early stages of system integration.

#### 4.1.1 Feedback Linearization for Simple Car Model

As a recap, the dynamics model for a simple car is given by

$$\begin{pmatrix} \dot{x} = u * \cos(\theta) \\ \dot{y} = u * \sin(\theta) \\ \dot{\theta} = \frac{u}{L} \tan(\phi) \end{pmatrix} \quad (4.1)$$

where the two inputs are  $[u, \phi]$  that represents the driving speed and steering angle. It is obvious from the above model that input and output are related in a nonlinear fashion. Differentiating the first two equations in 4.1 twice results in

$$\begin{pmatrix} \ddot{x} = \dot{u}_s \cos(\theta) - u_s^2 \dot{\theta} \sin(\theta) \\ \ddot{y} = \dot{u}_s \sin(\theta) + u_s^2 \dot{\theta} \cos(\theta) \end{pmatrix} \quad (4.2)$$

Define the virtual inputs as

$$\begin{pmatrix} \ddot{x} = v_1 \\ \ddot{y} = v_2 \end{pmatrix} \quad (4.3)$$

and the dynamic compensators as

$$\xi = u_1 \quad (4.4)$$

Also let

$$u_2 = \tan(u_\phi) \quad (4.5)$$

The resultant differentiated dynamics can be factorized as

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \dot{\xi} \cos(\theta) - \frac{1}{L} \xi u_s^2 \sin \theta \\ \dot{\xi} \sin(\theta) + \frac{1}{L} \xi u_s^2 \cos \theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\frac{1}{L} \xi u_s \sin(\theta) \\ \sin(\theta) & \frac{1}{L} \xi u_s \cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\xi} \\ u_2 \end{bmatrix} \quad (4.6)$$

solving the above system of two equations simultaneously for  $[\dot{\xi}, u_2]$  gives

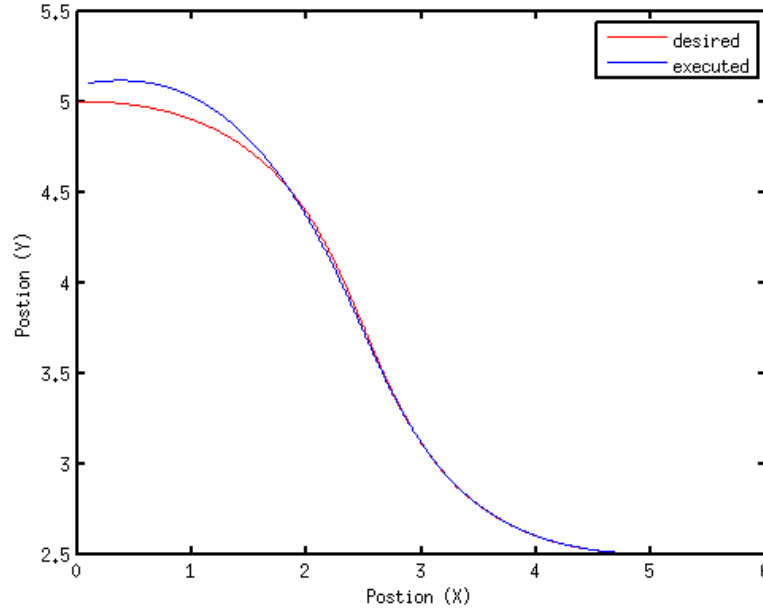
$$\begin{aligned} \dot{\xi} &= v_1 \cos \theta + v_2 \sin \theta \\ u_2 &= ((v_2 \cos \theta - v_1 \sin \theta) / \xi^2) \end{aligned} \quad (4.7)$$

let  $\vec{v} = [v_1, v_2]^T$  and  $\vec{y} = [x, y]^T$ , since  $\vec{v} = \ddot{\vec{y}}$ , a stable control law is given by

$$\vec{v} = \ddot{\vec{y}}_d - k_p(\vec{y} - \vec{y}_d) - k_d(\dot{\vec{y}} - \dot{\vec{y}}_d) \quad (4.8)$$

To see that the above control is stable, equation 4.8 can be rewritten in a state space form as

FIGURE 4.1: Feedback Linearization controller simulation result



$$\begin{bmatrix} \dot{\vec{y}} - \dot{\vec{y}}_d \\ \ddot{\vec{y}} - \ddot{\vec{y}}_d \end{bmatrix} = \begin{bmatrix} 0 & I \\ -k_p & -k_d \end{bmatrix} \begin{bmatrix} \vec{y} - \vec{y}_d \\ \dot{\vec{y}} - \dot{\vec{y}}_d \end{bmatrix} \quad (4.9)$$

for positive definite  $k_p$  and  $k_d$ , system 4.9 is globally asymptotically stable. And resulting virtual inputs can be mapped to physical inputs using equation 4.7

With above control, a simulation is conducted in Matlab using the simple car model. From start point  $X_0 = [0, 5, 0]$  to goal point  $X_f = [5, 2.5, 0]$ , the start and final velocity are zero. The desired trajectory is generated by fitting a quintic (fifth order) polynomial between the start and end points. Figure 4.1 below shows the simulation result.

## 4.2 Forward-Backward Sweep Method

Because what we are aimed at solving for is the local optimal trajectory and control, it can be reasonably assumed that for small time intervals no path constraint is to be imposed on the planner. Therefore the Forward-Backward Sweep Method (FBSM) [ ] [ ] can be a relatively easy to implement and effective way for solving such an optimization problem. Assume that the discrete nonlinear system dynamics is given by

$$x_{i+1} = fi(x_i, u_i), \quad (4.10)$$

and here we use a linear quadratic cost function of the form

$$\begin{aligned} J_0 &= L_N(x_N) + \sum_{i=0}^{N-1} L_i(x_i, u_i) \\ &= \frac{1}{2}x^T(N)P_f x(N) + \sum_{i=0}^{N-1} \frac{1}{2}(x_i^T Q x_i + u_i^T R u_i) \end{aligned} \quad (4.11)$$

and define the cost-to-go to be the above cost from current to the end ( $J_i = L_N(x_N) + \sum_i^{N-1} L_i(x_i, u_i)$ ). The goal is to locally optimize the Hamiltonian-Jacobi-Bellman (HJB) value function at each step defined by

$$V_i(x) = \min_{u_{i:N-1}} J_i(x, u_{i:N-1}) \quad (4.12)$$

and the above value at time  $i$  is written in a recursive form by

$$V_i(x) = \min_u Q_i(x, u) = \min_u \{L_i(x, u) + V_{i+1}(f_i(x, u))\} \quad (4.13)$$

where  $Q$  is the unoptimized value function. To minimize  $Q$  at step  $i$ , the perturbation around  $i$  is defined as

$$\Delta Q_i(\delta x_i, \delta u_i) = L_i(x_i + \delta x_i, u_i + \delta u_i) - L_i(x_i, u_i) + V_{i+1}(f(x_i + \delta x_i, u_i + \delta u_i)) - V_{i+1}(x_i, u_i) \quad (4.14)$$

Expanding the above equation to second order results in

$$\Delta Q_i \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta x_i \\ \delta u_i \end{bmatrix}^T \begin{bmatrix} 0 & \nabla_x Q_i^T & \nabla_u Q_i^T \\ \nabla_x Q_i & \nabla_x^2 Q_i & \nabla_{xu} Q_i \\ \nabla_u Q_i & \nabla_{ux} Q_i & \nabla_u^2 Q_i \end{bmatrix} \begin{bmatrix} 1 \\ \delta x_i \\ \delta u_i \end{bmatrix} \quad (4.15)$$

Given the principle of optimality, the choice of  $\delta u_i$  should minimize  $\Delta Q$ , in other words

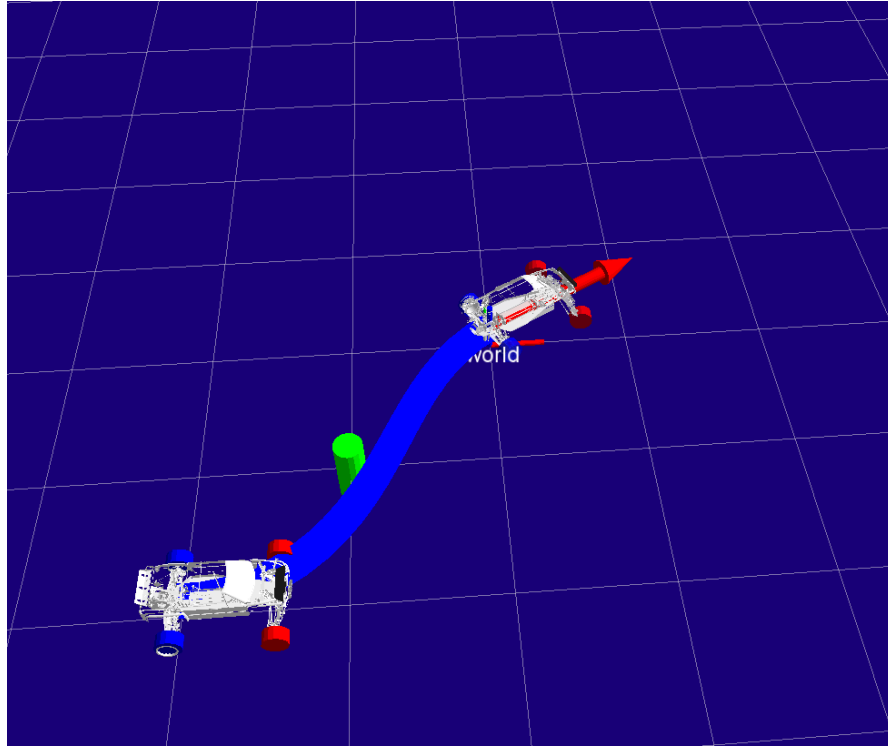
$$\frac{\partial \Delta Q}{\partial \delta u_i} = 0 \quad (4.16)$$

which gives the optimal  $\delta u$  as

$$\delta u_i^* = K_i \cdot \delta x_i + \alpha_i k_i \quad (4.17)$$



FIGURE 4.2: Feedback Linearization controller simulation result



where

$$K_i = -\nabla_u^2 Q_i^{-1} \nabla_{ux} Q_i \quad (4.18)$$

$$k_i = -\nabla_u^2 Q_i^{-1} \nabla_u Q_i \quad (4.19)$$

and  $\alpha$  is the step size chosen such that the controls result in a sufficient decrease in the value function.

With the above conclusions, a forward-backward process can be implemented to calculate the controls that yield locally optimal trajectory according to the designed cost function. The algorithm first steps backwards in time and recursively find  $K_i$  and  $k_i$  as defined in equations 4.18 and 4.19. Incorporating these values with the system dynamics, the controls are then updated forward in time. This forward backward process is repeated until convergence of the cost function. Details on the step-through of the algorithm is provided in Appendix C. This method is simulated in ROS with visualization results shown in figure 4.2. The vehicle in the back and front illustrates current and goal position respectively, and the blue line denotes the computed optimal trajectory.

# Chapter 5

## Results And Discussion

### 5.1 Experiment Description and Results

This chapter shows some of experimental results along with discussions of their significance. Given the floor map shown in Chapter 3, the vehicle is to autonomously navigate itself through the area while avoiding obstacles.

The path that the vehicle takes should also be locally optimal according to the calculation in the previous chapter. The relationship between the vehicle and world frame as well as the definition of frames within the vehicle is illustrated in Appendix B. It is necessary to not that since the brushless motor that drives the vehicle has a large minimal speed, while the AMCL method cannot provide an accurate estimate if robot runs at high speed

FIGURE 5.1: Computed vs. Actual Trajectory in Controller Testing

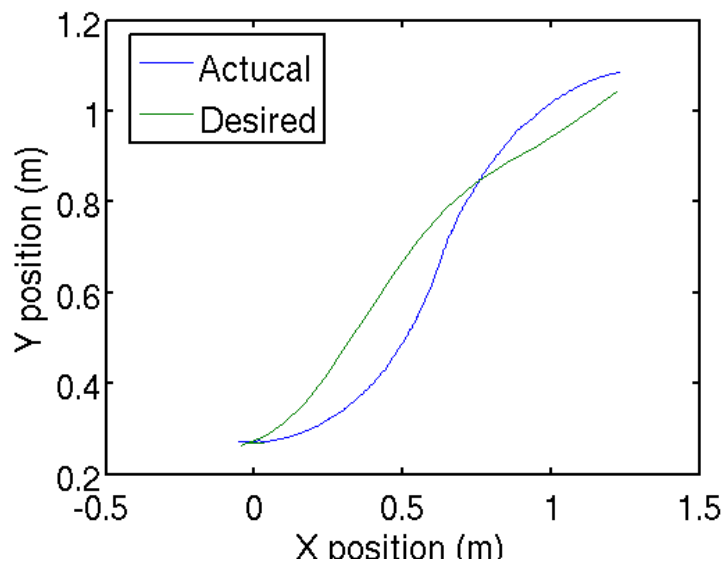


FIGURE 5.2: Video Screenshot



(state update is not able to catch up with actual position change). Therefore parameter optimization is limited on control of velocity input signal by setting very small  $R$  in the cost function given by 4.11. Figure 5.1 shows the FBSM computed optimal/desire trajectory compared to the actual executed trajectory record by motion capture in an in-lab calibration test. We can see that the overall trend is correct and the discrepancy is within a relatively small range (max around  $20cm$ ). The short distance traveled is due to limited lab space.

Global trajectory is pre-selected as the red dots showing in the figure 5.2. The goal speed was assumed constant. Blue line in the same figure shows local optimal trajectory computed using the Direct Sweep Method. The algorithm uses the vehicle's current location as the initial position and last step speed as initial speed. Final position is the 5th closet point in global trajectory in the direction of the current heading. The result demonstrates that mobile robot was able to track the trajectory for 10 loops at  $1 m/s$ .

The video compares real world vehicle view (recorded using the onboard Kinect) with the Rviz visual environment (with the graphics generated by sensor and command messages). The Video in the attachment shows the entire process of loop trajectory optimization and tracking. Note that the vehicle is manual stopped on the end off the hallway due the internet connection lose.

## 5.2 Future Work

Although odometry is not suppose to give perfectly accurate localization. But improving odometry result over time will certainly result in a better localization performance. With a better odometry model, the number of particles used for localization can be reduced which gives a faster computational time and a higher rate of state update.

The magnetometer is there to provide a constant reference (magnetic north) which compensates for the drift of the gyroscope. Therefore fusing the gyro and magnetometer

information can greatly enhance the accuracy of odometry. There is an IMU(Inertial measurement unit) built into the Arduino Auto pilot board, which in addition to the gyro contains a magnetometer as well as an accelerometer. But with the current hardware set up, several magnets are used for the battery case cap which affects the magnetometer measurement dramatically. Therefore future design can include removing sources of magnetic interference and incorporating additional sensors for noise removal and higher accuracy orientation estimation.

The orientation of the vehicle ranges for from  $-\pi$  to  $\pi$  and the solver computes trajectory only in this range. The resulting issue that arises is shown in the figure below. With a starting heading of  $\theta_0 = -3rad$  and a goal of  $\theta_f = 3.1rad$ , the solver will calculate optimal control input as turning in the direction that crosses zero (since mathematically there's only one direction to go from -3 to 3.1 on the real line). But simply by adding a  $2\pi$  wrap-around doesn't completely solve the problem. This issue renders the optimizer to give suboptimal trajectories in situations like this and is worth the effort to look at in the future.

FIGURE 5.3: issue with trajectory



## Chapter 6

# Conclusion

This essay aims at introducing a complete system integration process for high speed/low cost autonomous navigation testbeds from hardware to software and finally choice of perception and control schemes. Limitations on both the hardware processing power and software computational time restricts how high a speed the vehicle can accurately and reliably navigate itself. The proposed system uses a collection of standard off-the-shelf mechanical and electrical components. At the time of experimentation, the onboard computer used is not powerful enough to handle the localization and control algorithms, so this part is done on a host laptop and onboard computer handles only collection of sensor data and execution of command given by the host computer. Communication is established via the Wifi network and a large restriction on the overall performance is imposed by communication speed and reliability (drop of packets, lost connection, etc). Therefore it is worthwhile to try and migrate all computation onto the vehicle which makes the system more closely meet the standard of full autonomy. The choice of *gmapping* as our SLAM method due to its ability to draw particles in a more accurate way and thus dramatically reduces the required number of samples and unnecessary resampling actions. For our online localization algorithm, AMCL is chosen for its simplicity in terms of implementation as well as adequate perform for our purposes. The Forward-Backward Sweep Method is used as our locally trajectory planning and tracking scheme given its effectiveness in generating locally optimal trajectories while also providing a feedback control law. Overall the proposed system has achieved stable and continuous autonomous navigation circumventing the close loop hallway as described in chapter 3 with an average speed of 1 *m/s*. Much room exists for discussion and improvement, but this essay serves as a detailed guidance in implementing related systems as well as considerations necessary in the choice of navigation schemes.

# Appendix A

## Experiment procedure

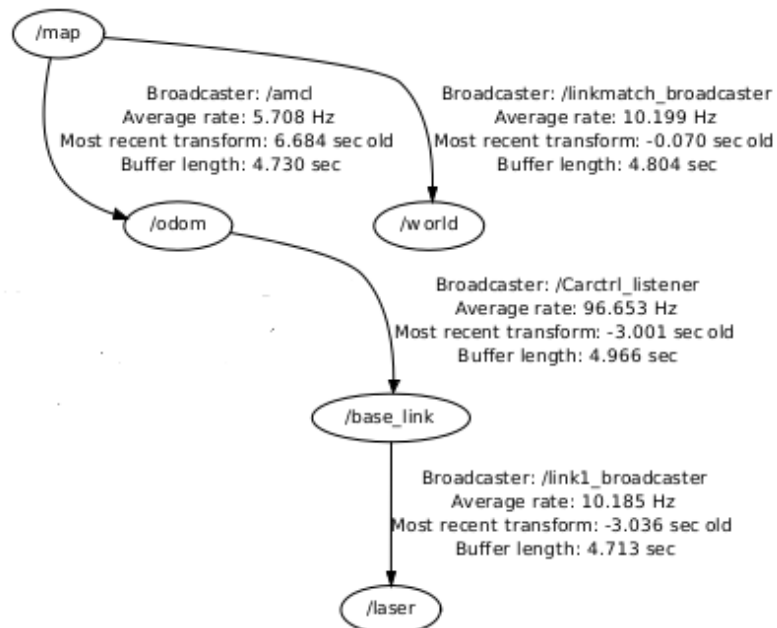
1. Set-up system hardware and joystick control
2. Feedback linearization control using motion capture to localize
3. sweep method and dynamic programming control using motion capture localize
4. Kinetic visual slam using motion capture fake odometry
5. Slam using laser sensor and fake odometry
6. slam gmapping with kinematic model odometry
7. Integration of Slam and differential dynamic programming

## Appendix B

# System Frames

The following are detailed diagrams showing the correlation of different frames used by the system, the node that publishes them (broadcaster) as well as their rate of broadcast. This information is important in terms of verification that the system is software-wise integrated properly and is very useful for debugging purposes.

FIGURE B.1: Position of The Vehicle Relative to The World

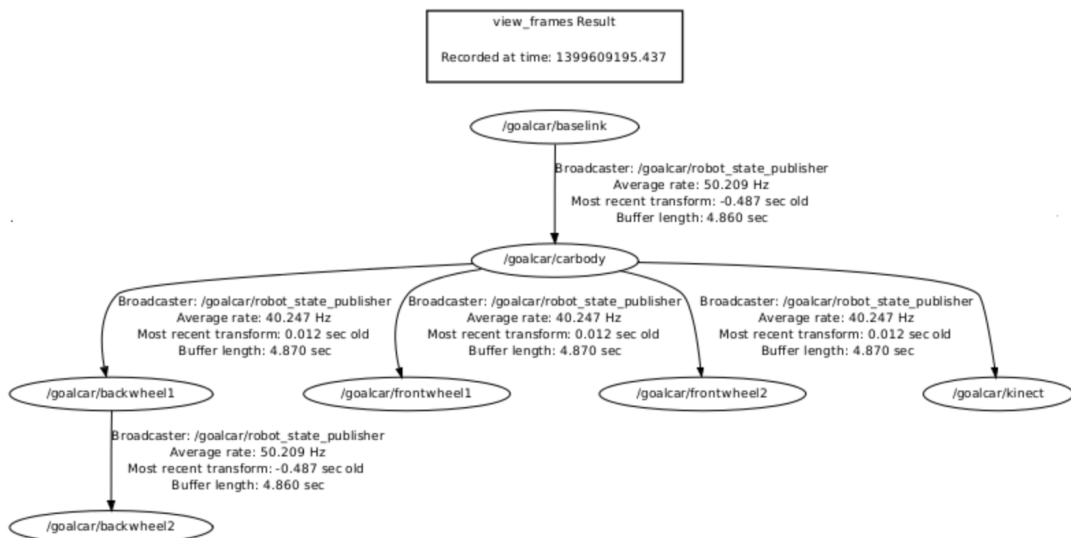


Appendix A. List of Project step

FIGURE B.2: Vehicle Body Frames



FIGURE B.3: system frames





## Appendix C

# Forward Backward Sweep Method

As mentioned in Chapter 4, the *backwards sweep* finds the coefficients  $K_i$  and  $k_i$  as defined in equations 4.18 and 4.19. Detailed steps are given by (according to [5])

$$V_x = \nabla L_N \quad (\text{C.1})$$

$$V_{xx} = \nabla_x^2 L_N \quad (\text{C.2})$$

$$k = N - 1 \rightarrow 0 \quad (\text{C.3})$$

$$Q_x = \nabla_x L_i + A_i^T (V_{xx}) A_i \quad (\text{C.4})$$

$$Q_u = \nabla_u L + B^T V_x \quad (\text{C.5})$$

$$Q_{xx} = \nabla_x^2 L_i + A_i^T (V_{xx}) A_i \quad (\text{C.6})$$

$$Q_{uu} = \nabla_u^2 L + B_i^T (V_{xx}) B_i \quad (\text{C.7})$$

Appendix C. *List of Project step*

$$Q_{ux} = \nabla_u \nabla_x L_i + B_i^T (V_{xx}) A_i \quad (\text{C.8})$$

$$k_i = -Q_{uu}^{-1} Q_u \quad (\text{C.9})$$

$$K_i = -Q_{uu}^{-1} Q_{ux} \quad (\text{C.10})$$

$$V_x = Q_x + K_t^T Q_u \quad (\text{C.11})$$

$$V_{xx} = Q_{xx} + D^T Q_{ux} \quad (\text{C.12})$$

While the *forward sweep* updates the controls by

$$\delta x_0 = 0, \quad (\text{C.13})$$

$$V_0' = 0 \quad (\text{C.14})$$

$$k = 0 \rightarrow N - 1 \quad (\text{C.15})$$

$$u_i' = u_i + \alpha k_i + K_i \delta x_i \quad (\text{C.16})$$

$$x_{i+1}' = f_i(x_i', u_i') \quad (\text{C.17})$$

$$\delta x_{i+1} = x_{i+1}' - x_{i+1} \quad (\text{C.18})$$

$$V_o' = V_o' + L_i(X_i', u_i') \quad (\text{C.19})$$

$$V_o' = V_0' + L_N(x_N') \quad (\text{C.20})$$

# Bibliography

- [1] G. Grisetti, C. Stachniss, and W. Burgard. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 2007.
- [2] G. Grisetti, C. Stachniss, and W. Burgard. Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In *ICRA Robots and Automation*, 2005.
- [3] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2006.
- [4] Dieter. Fox. Adapting the Sample Size in Particle Filters Through KLD-Sampling. *International Journal of Robotics Research*, 22, 2003.
- [5] M. Kobilarov. EN530.603 Applied Optimal Control EN530.603 Applied Optimal Control. Lecture 9: Numerical Methods for Deterministic Optimal Control, October 2013.

## *Biographical Statement*

Ying Xu was born July 5th,1989 in Wuhan, China. She completed her undergraduate studies at Iowa State University in Mechanical Engineering. After that she started her graduate work at Johns Hopkins University, Baltimore obtained he MSE in Mechanical Engineering under the Robotics Track in 2014.