ME3243 Project 1

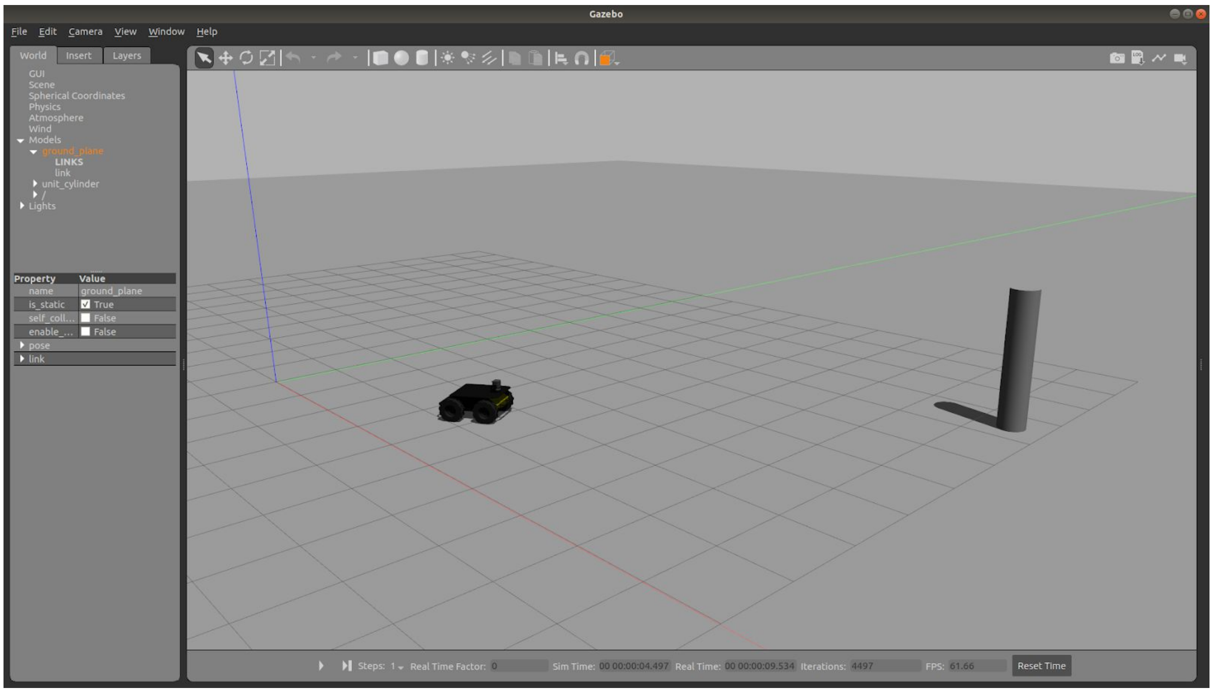
Tuning PID Controller

20 September 2020

Ng Wei Jie, Brandon A0184893L

Summary

The task is to control a Husky robot in a ROS environment with PID controllers. The Husky robot starts from an initial position, moves towards the pillar, which is located at a certain position, and stops at a predetermined distance from the pillar. The PID controllers for steering control and motor control are manually configured to achieve a movement that is smooth, accurate, and fast. The project's learning objective is to understand the implementation and turning process of PID gains using the ROS simulation environment Gazebo as shown in Figure 1.



Property	Value
name	ground_plane
is_static	✓ True
self_coll…	False
enable_…	False

Steps: 1 Real Time Factor: 0 Sim Time: 00 00:00:04.497 Real Time: 00 00:00:09.534 Iterations: 4497 FPS: 61.66

Figure 1: ROS simulation environment Gazebo

Initial Settings

The project uses ROS Melodic to tune the PID gains of a 4 wheel mobile robot Husky. The ROS simulation environment is Gazebo. The PID gains are declared in the config.yaml and the values are adjusted to obtain a set of gains that allow the robot to achieve a smooth, accurate, and fast movement. The linear and angular movements of the robot are assumed to be independent of each other as stated in the lecture slides.

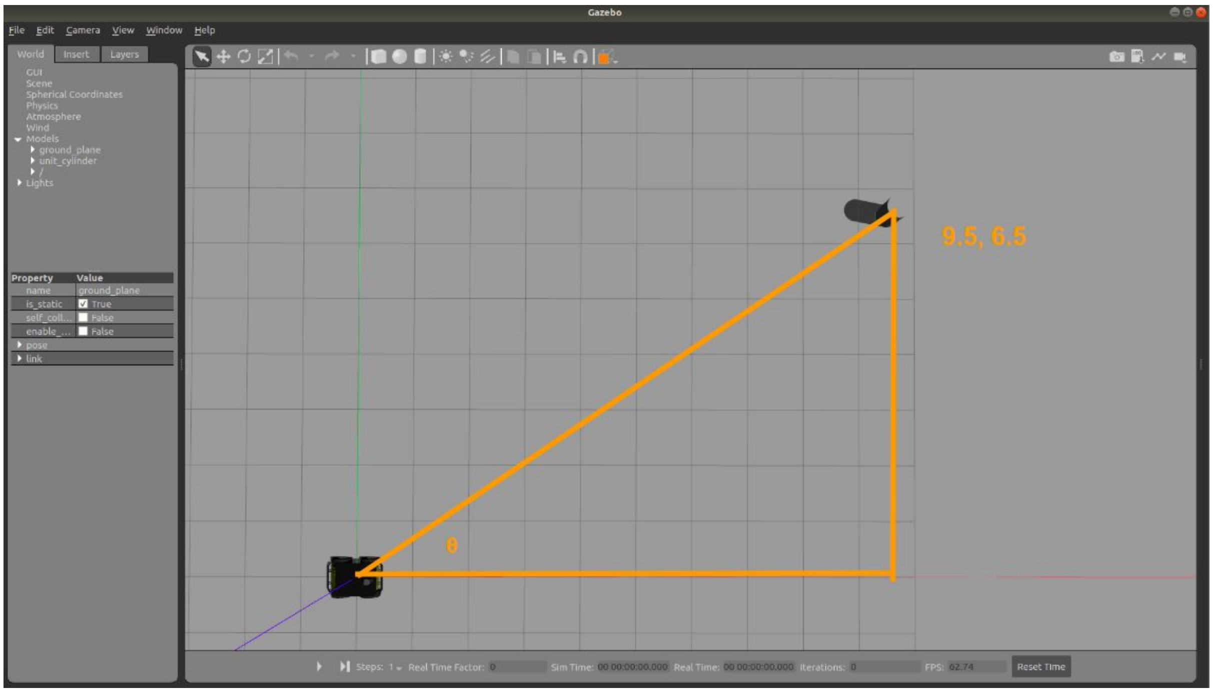


Figure 2: Initial settings of robot

The pole location is according to the last two digits of the matriculation number with digits xy. The pillar location is obtained with $(5 + \frac{x}{2},\ 5 + \frac{y}{2})$. As my last two digits are $x = 9$ and $y = 3$, the pillar location is at $(5 + \frac{9}{2},\ 5 + \frac{3}{2}) = (9.5,\ 6.5)$.

The predetermined distance from the pillar at which the robot stops is according to the last digit of matriculation number with digit y. The distance is obtained with $(1 + \frac{y}{10})$. As my last digit is 3, distance is $(1 + \frac{3}{10}) = (1.3)$.

With the robot at origin facing the positive x direction in Figure 2, the initial distance between the Husky robot and the pole is $\sqrt{9.5 - 0^2 + 6.5 - 0^2} \approx 11.51$ using Euclidean distance formula. The initial orientation of the pole with respect to the Husky robot is $tan^{-1}\frac{6.5}{9.5} \approx 34.4^{\circ} = 0.600rad$ with basic trigonometry functions.

Evaluation Of PID

PID controller uses a closed loop control feedback to keep the actual output of the system as close to the expected output as possible. To understand characteristics of a good or bad PID controller, an intuition of what individual PID gains do as shown in Table 1.

Table 1: Effect of PID gains[1]

Gain	Effect
Kp	Contributes to stability, medium-rate responsiveness.
Ki	Tracking and disturbance rejection, slow-rate responsiveness. May cause oscillations.
Kd	Fast rate responsiveness. Sensitive to noise.

Table 2 shows the effect of changing the PID gain on the performance of the system. The PID gains are evaluated in terms of rise time, overshoot, settling time, and steady state error.

Table 2: Effect of PID parameters on performance

Parameter Increase	Rise Time	Overshoot	Settling Time	Steady State Error
Kp	Increase	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

The error vs time graphs of a poorly tuned PID for motoring control (left) and steering control (right) is shown in Figure 3. The error shown in Figure is on the more extreme end of a poorly tuned PID. The corresponding gains for the poorly tuned PID gains are shown in Figure 4.

[1] https://www.youtube.com/watch?v=Mk1ygHj4zxw&t=332s

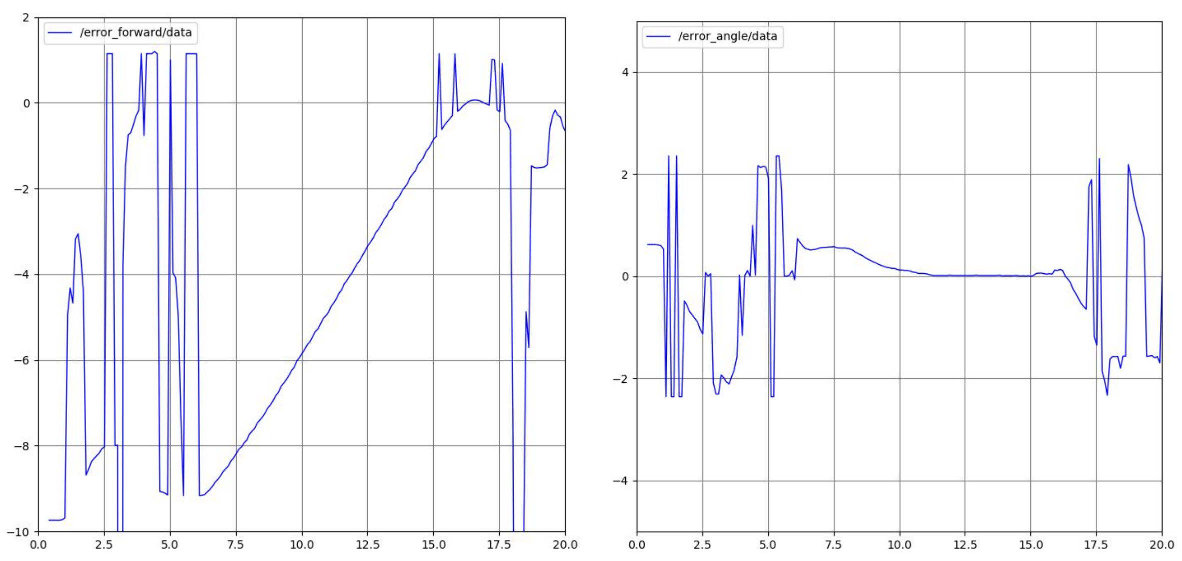
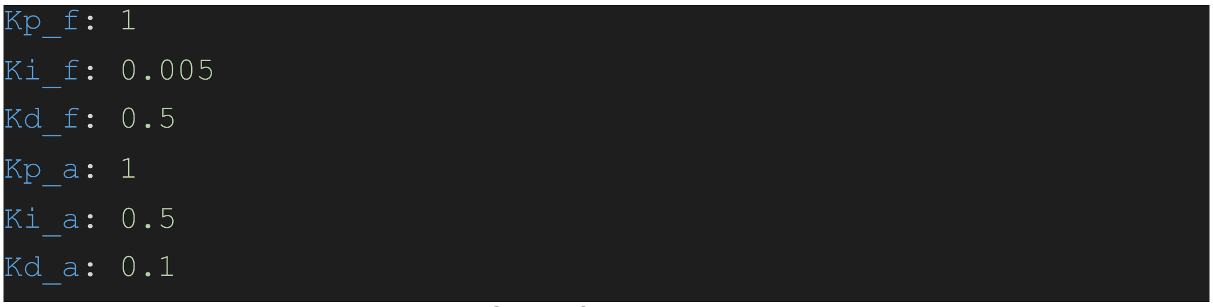


Figure 3: Error vs time graphs of poorly-tuned PID



```
Kp_f: 1
Ki_f: 0.005
Kd_f: 0.5
Kp_a: 1
Ki_a: 0.5
Kd_a: 0.1
```

Figure 4: Gains for poorly-tuned PID

The following description on the characteristics of the graph shows how this set of PID gains is poorly tuned. From Figure 3, it is hard to pinpoint the problems with gains in the motor control when the steering control is very problematic.

1. The angle error overshoots by a large margin when the robot starts moving. The robot is making sharp circular turns, and the movement is not smooth. The Kp for steering control might be too large, resulting in large control action and sharp turns at the beginning of the movement.
2. The angle error and forward error did not achieve steady state error at 15 seconds when the robot reaches the destination but oscillates wildly. The Kd for steering control might be too large, and the derivative of the error signal is causing the robot to overreact to the noise.

The error vs time graphs of a better but not well-tuned PID for motoring control (left) and steering control (right) is shown in Figure 5. The corresponding gains for the not well-tuned PID are shown in Figure 6.

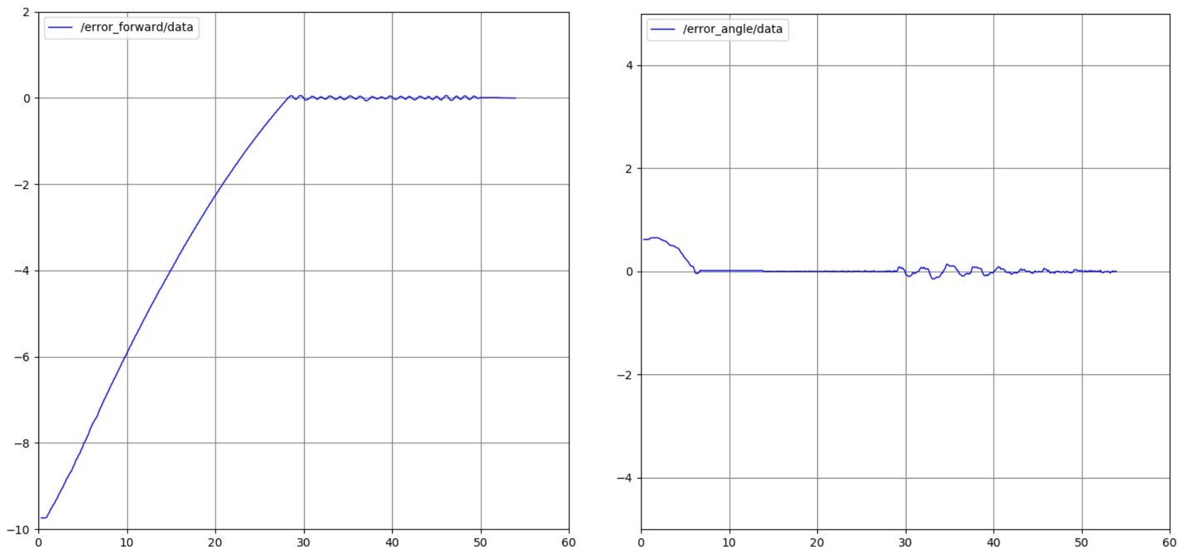
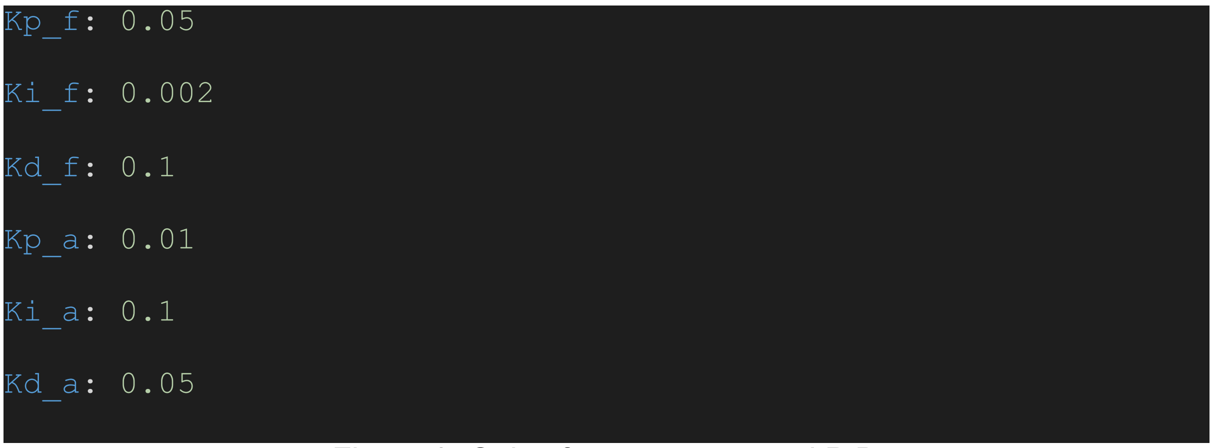


Figure 5: Error vs time graphs of not well-tuned PID



```
Kp_f: 0.05

Ki_f: 0.002

Kd_f: 0.1

Kp_a: 0.01

Ki_a: 0.1

Kd_a: 0.05
```

Figure 6: Gains for not well-turned PID

The following description on the characteristics of the graph shows how this set of PID gains is not well-tuned despite being better than the previous one in Figure 3.

1. The forward error's rise time is about 28 seconds which is very long. The movement is slow. The Kp for motor control might be too low and, therefore, the robot has lower control action to drive itself towards the target position.

2. The forward error’s settling time is 50 seconds which is very long. The movement is slow. The Kd for motor control might be low and increasing it might be able to help reduce the settling time.
3. The forward error overshoots and causes the robot to move back and forth at the destination. The movement is not smooth. The Ki for motor control might be a little too large. This results in small oscillations before settling down.
4. The angle error starts to oscillate at 30 seconds even after steady state error is driven to zero. The steering is not smooth and the angle error has a long settling time. The Kd for steering control might be too large, and the derivative of the error signal is causing the robot to overreact to the noise.
5. The angle error did not slowly reduce in magnitude as the magnitude approaches zero and causes the robot to swerve suddenly. The steering is not smooth. This happens likely due to the low Kp for steering control that causes the robot to not be responsive to the error at the start. Until the error in angle error starts accumulating later on, the Ki starts to drive the steady state error to zero.

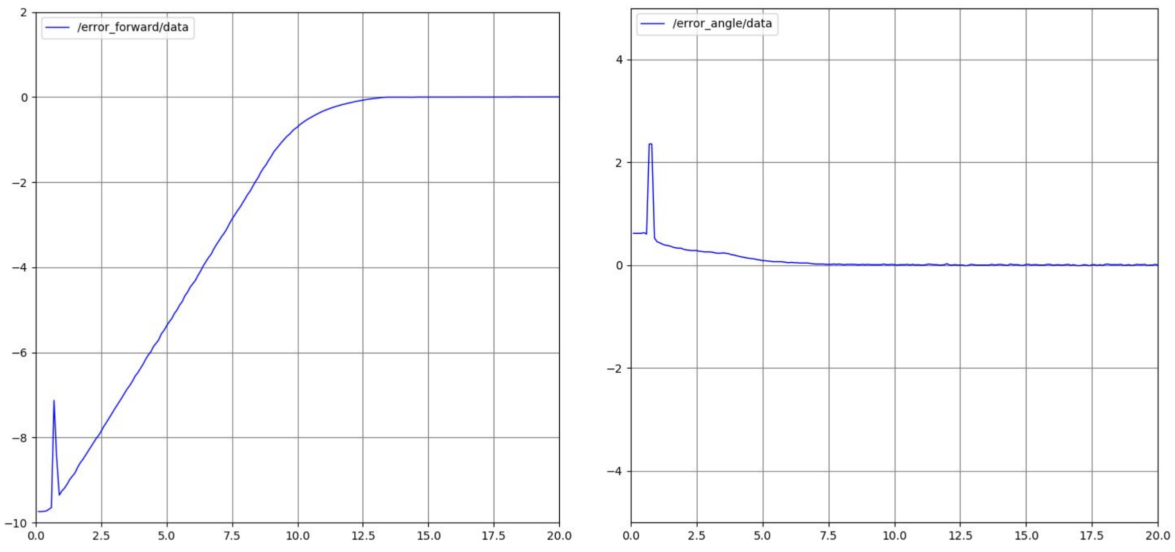
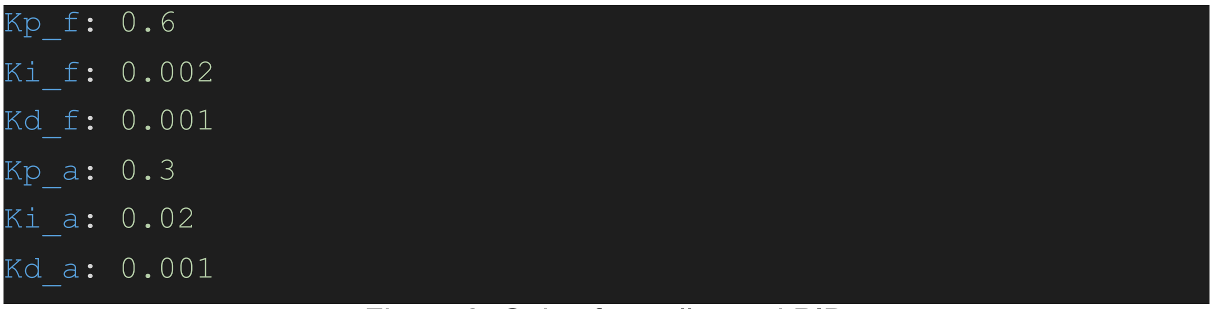


Figure 7: Error vs time graphs of well-tuned PID

The error vs time graphs of well-tuned PID for motoring control (left) and steering control (right) is shown in Figure 7 below. The corresponding gains for the well-tuned PID are shown in Figure 8.



```
Kp_f: 0.6
Ki_f: 0.002
Kd_f: 0.001
Kp_a: 0.3
Ki_a: 0.02
Kd_a: 0.001
```

Figure 8: Gains for well-tuned PID

The following description on the characteristics of the graph shows how this set of PID gains is well-tuned as compared to the previous two.

1. The forward error achieves a faster settling time and rise time of less than 15 seconds. Kp for motor control is much larger and provides more control action to the robot. Therefore, the movement is much faster.
2. The forward error slowly reduces in magnitude as the magnitude approaches zero. This movement is smooth and the robot gently stops. The derivative of the error signal in Kd for motor control helps the robot achieve a faster initial response and a slower response towards the end.
3. The forward error did not overshoot and there is no oscillation. The movement is smooth and the robot is not swaying unnecessarily. Kp and Ki for motor control are not too large and a good value of Kd for motor control helps to dampen the system.
4. The angle error did not overshoot and slowly reduced in magnitude as the magnitude approaches zero. The steering is smooth. The Ki and Kp for steering control are not too large to induce oscillations due to violent steering.

5. The angle error achieves a faster settling time and rise time of less than 12 seconds. Kp for steering control is sufficiently large and provides more control action in the steering. Ki is also sufficiently large to drive the angle error to zero as the error accumulates with time. Hence, the steering is much faster.

Process Of Tuning PID

The ideal gains for PID steering and motor controllers are shown in Figure 8 and are determined experimentally. However, understanding how the performance and inner workings of a P, PI, and PD is also important in forming the thought process of tuning the ideal PID gains. This is done by setting the relevant gains to 0.

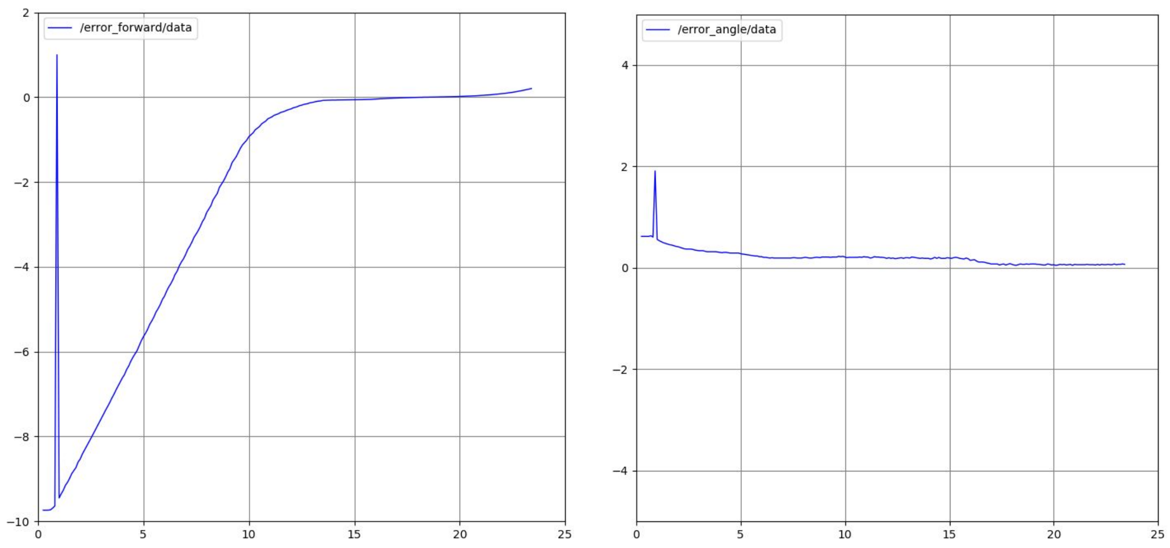
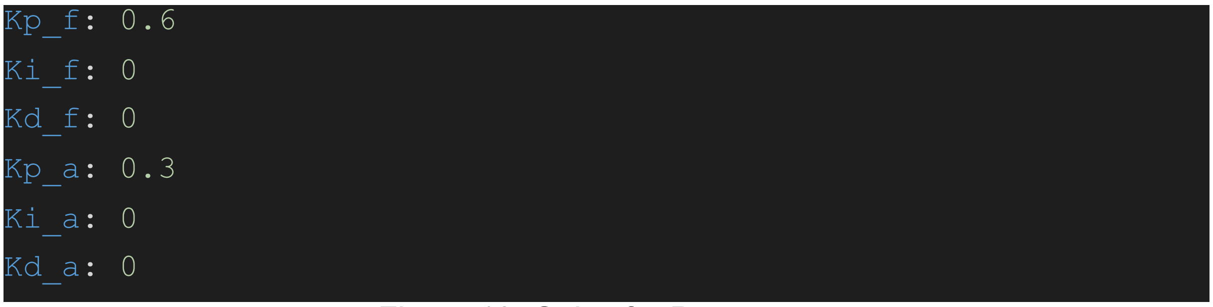


Figure 9: Error vs time graphs for P controller



```
Kp_f: 0.6
Ki_f: 0
Kd_f: 0
Kp_a: 0.3
Ki_a: 0
Kd_a: 0
```

Figure 10: Gains for P controller

Figure 9 shows the error vs time graphs for P motor controller (left) and steering controller (right). The corresponding gains are shown in Figure 10. Observe that the steady state error for the angle is not zero even though the robot reaches the target at 15 seconds. The slight steering in the robot causes the robot to go off in forward error and then start oscillating or ramming into the pillar.

Figure 11 shows the error vs time graphs for PI motor controller (left) and steering controller (right). The corresponding gains are shown in Figure 12. Observe that introducing Ki to the steering controller, the steady state error for the angle is driven to zero before the robot reaches the target. The rise time and settling time for angle error is faster. In addition, notice that introducing Ki to the motor controllers, the rise time and settling time for forward error is noticeably faster by 3 seconds as compared to the P controller.

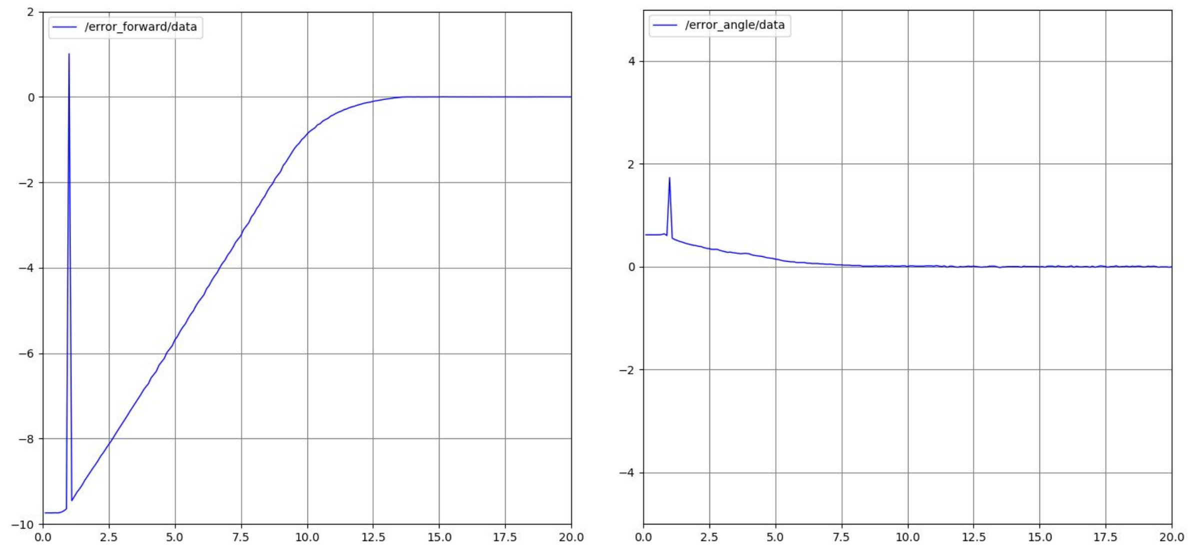
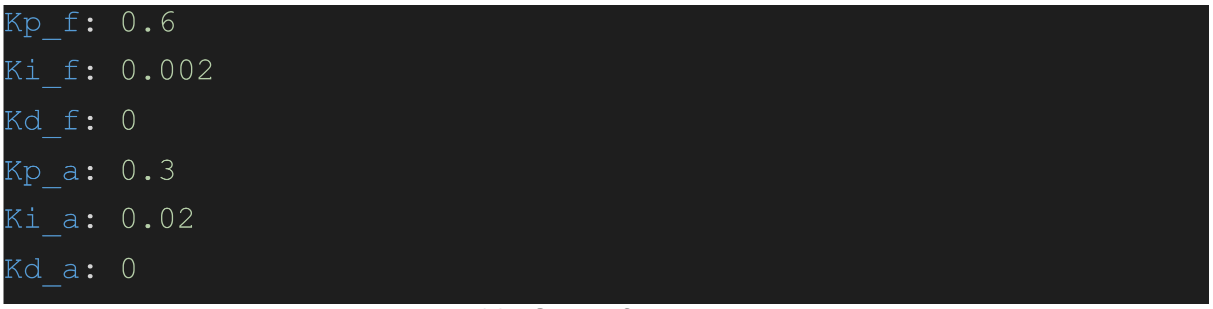


Figure 11: Error vs time graphs for PI controller



```
Kp_f: 0.6
Ki_f: 0.002
Kd_f: 0
Kp_a: 0.3
Ki_a: 0.02
Kd_a: 0
```

Figure 12: Gains for PI controller

Figure 13 shows the error vs time graphs for PD motor controller (left) and steering controller (right). The corresponding gains are shown in Figure 14. Observe that without Ki to the steering controller, the steady state error for the angle is not driven to zero unlike the PI controller. The same issue occurs in the P controller where a slight steering in the robot causes the robot to go off in forward error and then start oscillating or ramming into the pillar. In addition, notice that introducing Kd to the motor controllers, the rise time and settling time for forward error is longer by 1 second as compared to the P controller.

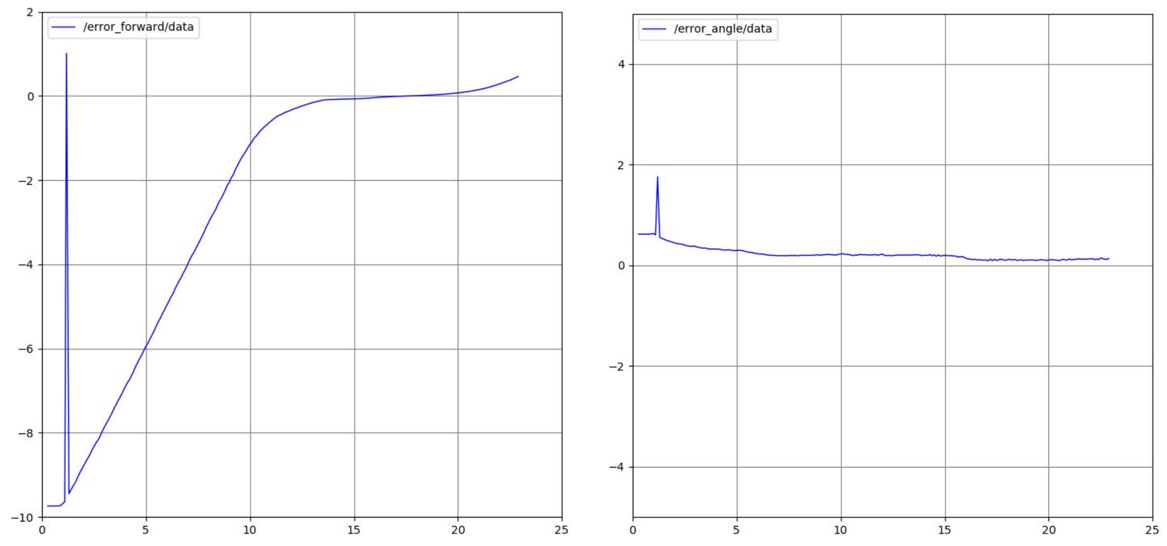
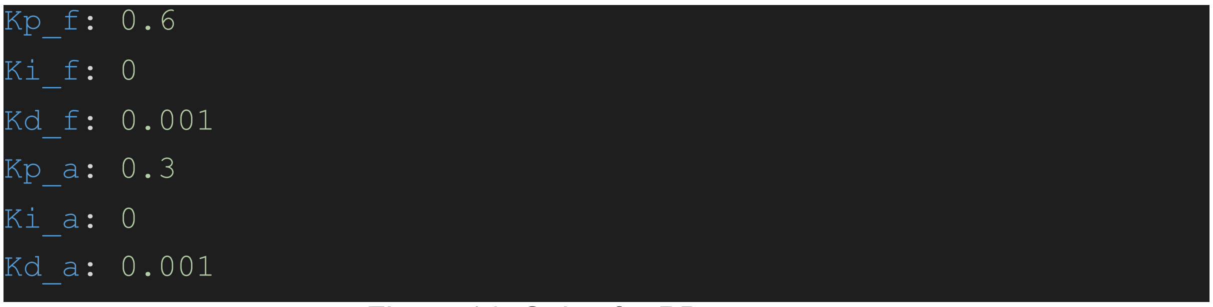


Figure 13: Error vs time graphs for PD controller



```
Kp_f: 0.6
Ki_f: 0
Kd_f: 0.001
Kp_a: 0.3
Ki_a: 0
Kd_a: 0.001
```

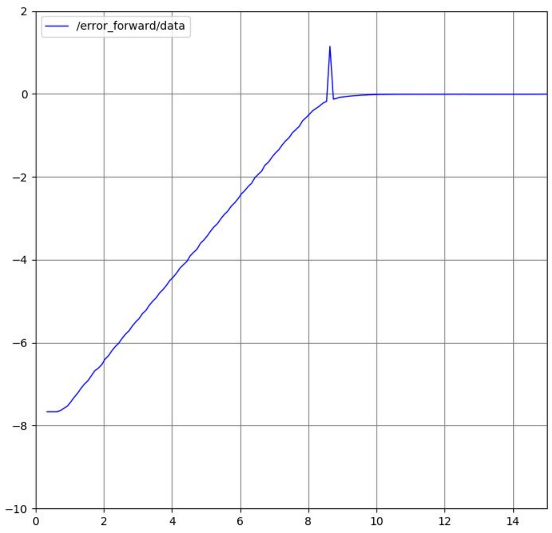
Figure 14: Gains for PD controller

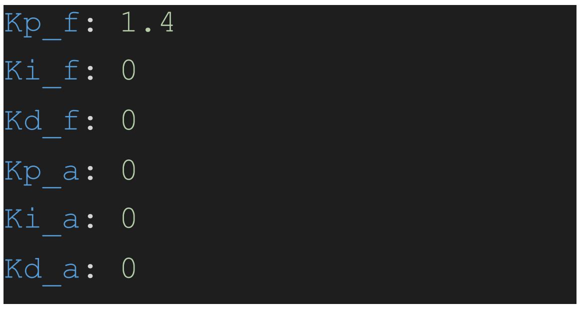
There are PID gains for motor control and steering control to be manually configured. There is a recommended guide to tune PID on the wikipedia page under the section manual tuning.[2] The gain for motor control is configured first with the pillar offset at 0. The steering control is configured thereafter.

1. Increase Kp for motor control until the forward error oscillates as shown in Figure 15.

[2] https://en.wikipedia.org/wiki/PID_controller

2. Half the Kp for motor control approximately to allow a “quarter amplitude decay” type response. The idea behind the quarter-amplitude damping is to eliminate error in the forward error as the controller responds so fast and overshoots the target output, resulting in oscillations.

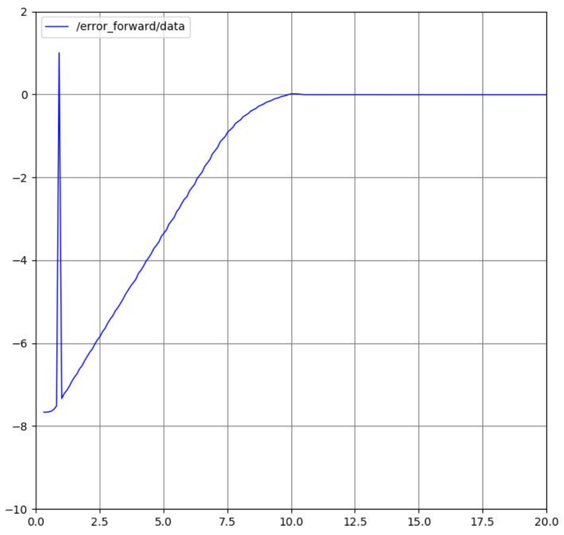


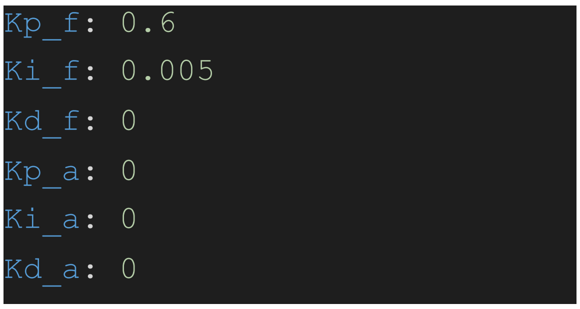


```
Kp_f: 1.4
Ki_f: 0
Kd_f: 0
Kp_a: 0
Ki_a: 0
Kd_a: 0
```

Figure 15: Configuring Kp for motor control

3. Increase Ki until offset is corrected. There is not much steady state error to begin with. A gain for Ki for motor control shown in Figure 16 is set before oscillations occur.

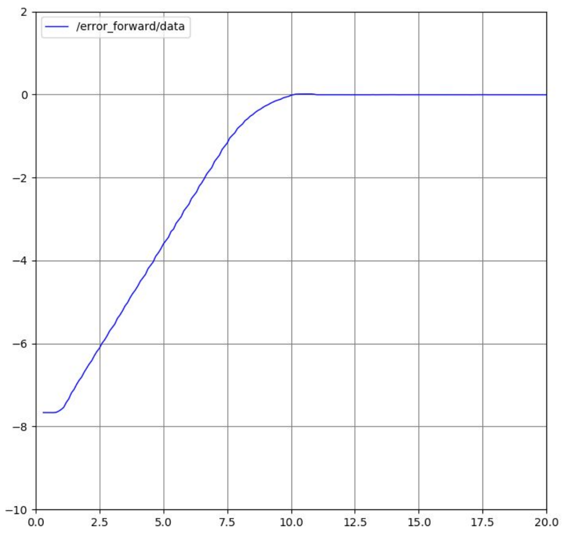


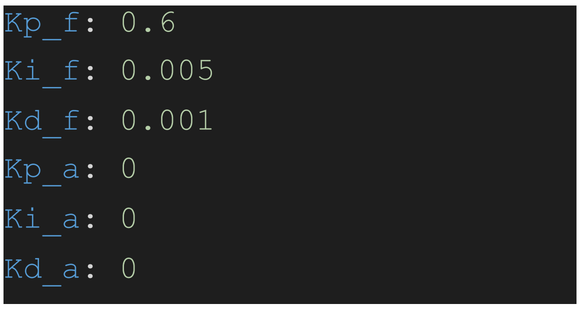


```
Kp_f: 0.6
Ki_f: 0.005
Kd_f: 0
Kp_a: 0
Ki_a: 0
Kd_a: 0
```

Figure 16: Configuring Ki for motor control

4. Increase Kd for motor control if required but not too much because it is susceptible to noise. A gain for Kd for motor control shown in Figure 17 is set to help the robot achieve a slower response at the end of the path to halt gently at the target.

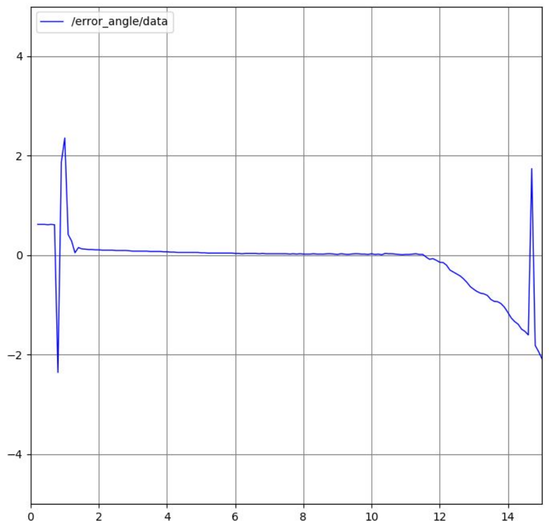


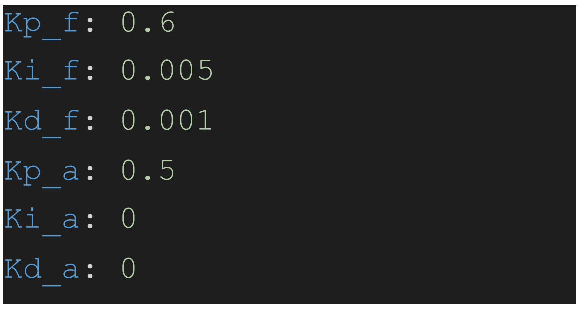


```
Kp_f: 0.6
Ki_f: 0.005
Kd_f: 0.001
Kp_a: 0
Ki_a: 0
Kd_a: 0
```

Figure 17: Configuring Kd for motor control

5. Increase Kp for steering control with the pillar at the correct y position until the angle error oscillates as shown in Figure 18.

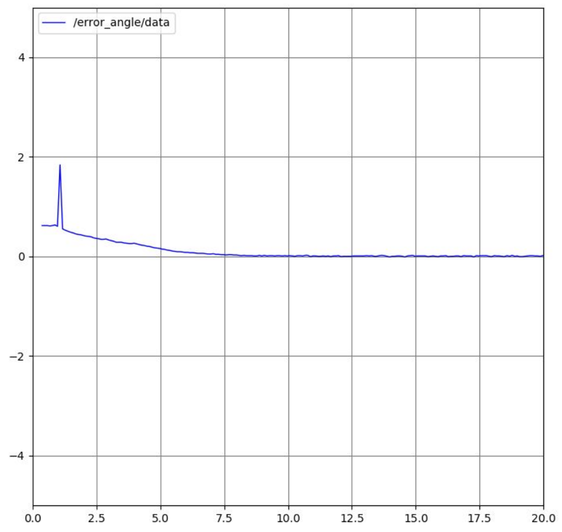


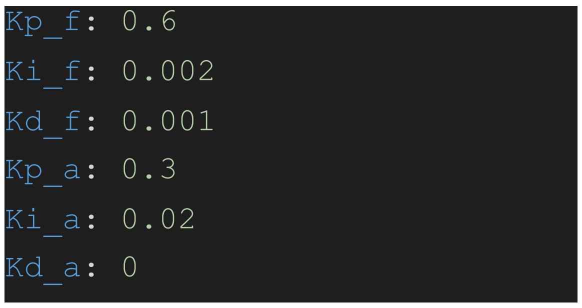


```
Kp_f: 0.6
Ki_f: 0.005
Kd_f: 0.001
Kp_a: 0.5
Ki_a: 0
Kd_a: 0
```

Figure 18: Configuring Kp for steering control

6. Half the Kp for steering control approximately to allow a “quarter amplitude decay” type response. Notice that the error angle is slow to correct with a lower Kp. Increase the Ki for steering control to correct the steady state error. Note that Ki for motor control is reduced as shown in Figure 19 as the forward error oscillates for this set of gains.





```
Kp_f: 0.6
Ki_f: 0.002
Kd_f: 0.001
Kp_a: 0.3
Ki_a: 0.02
Kd_a: 0
```

Figure 19: Configuring Kp for steering control

7. Increase Kd for steering control if required but not too much because it is susceptible to noise. A gain for Kd for motor control shown in Figure 7 is set to help the robot achieve a slower response at the end of the path to halt gently at the target.

Description Of ROS Simulation

The RQT graph displays the nodes and topics that are present in the ROS simulation with Gazebo as shown in Figure 20. Nodes are represented as ovals and topics are represented as rectangles.

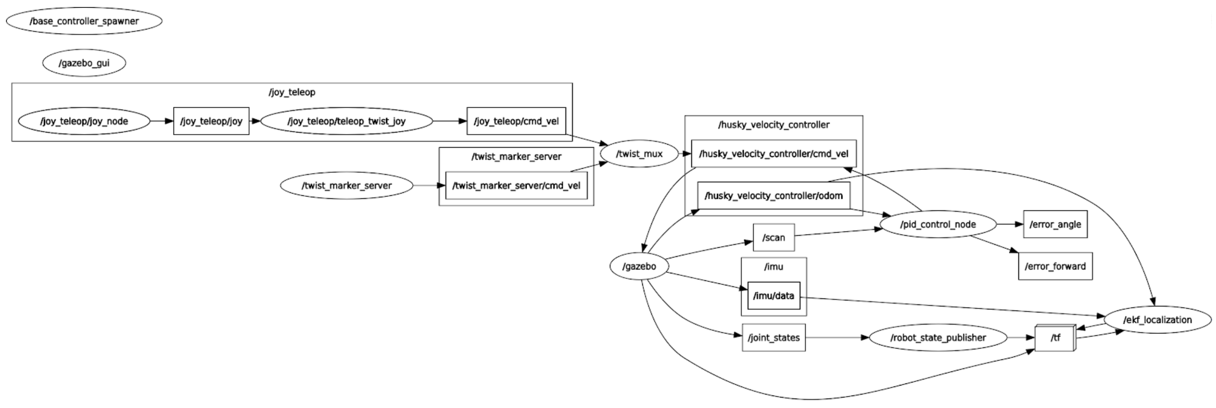


Figure 20: RQT graph of all nodes and topics in Gazebo simulation

The section of the RQT graph in Figure 21 shows the relevant nodes and topics for the PID control. The relevant nodes are /gazebo and /pid_control_node. The topics are /scan and /husky_velocity_controller/cmd_vel.

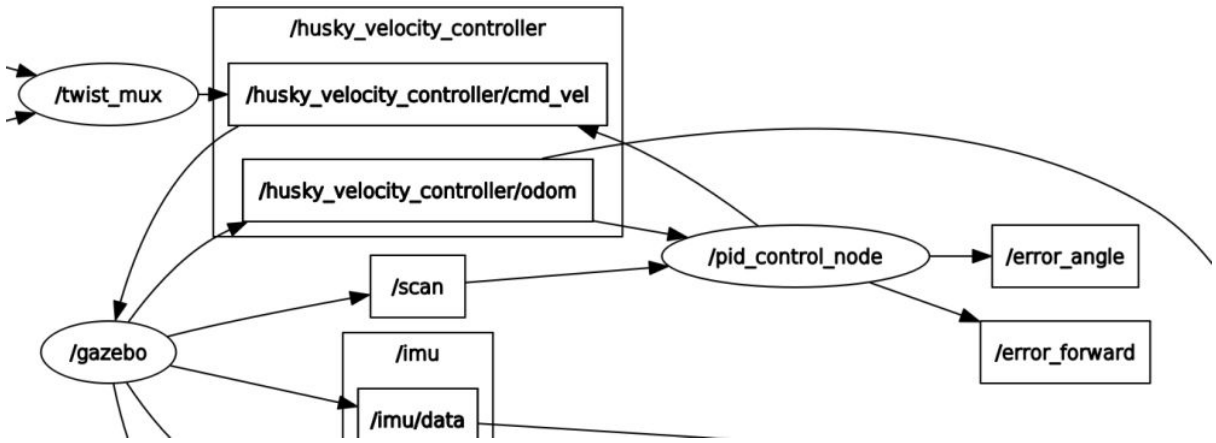

/husky_velocity_controller
/twist_mux
/husky_velocity_controller/cmd_vel
/husky_velocity_controller/odom
/pid_control_node
/error_angle
/scan
/error_forward
/gazebo
/imu
/imu/data

Figure 21: RQT graph of nodes and topics for PID control

Node: /gazebo

The node /gazebo is responsible for setting the husky robot and environment for simulation. Based on the forces applied onto the robot in the simulation, it provides odometry and 2D LIDAR reading as shown in Figure 21 where the node /gazebo is publishing the odometry through the topic /husky_velocity_controller/odom and the 2D LIDAR through the topic /scan. In the PID controller, only the topic /scan is used. The topic /husky_velocity_controller/odom is subscribed but not used by the node /pid_control_node for the PID algorithm.

Node: /pid_control_node

The node /pid_control_node is responsible for calculating and publishing the linear and angular velocity of the robot to the topic /husky_velocity_controller/cmd_vel. It calculates the linear and angular velocity using PIDs for motor control and steering control. The PIDs take in inputs relative angle and distance between the robot and husky. Information about angle and distance are obtained from the topic /scan. The node /pid_control_node forms the closed loop control system using feedback (relative angle and distance between the robot and husky) to control processed outputs (linear velocity and angular velocity) of the system.

Topic: /scan

The topic /scan provides 2D LIDAR measurements of the simulated environment. It has the message type sensor_msgs::LaserScan. As shown in the algorithm in the node /pid_control_node in Figure 22, the algorithm loops through the attribute ranges, which contains a list of distance measurements, to obtain the angle and distance of smallest distance. The relative angle and distance between robot and pillar are calculated as inputs to the PID controller.

```
for(int i = 0; i<arr_size; i++){
   if(scan_data_[i] < smallest_dist) {
       smallest_dist = scan_data_[i];
                       scan_ang_   =   scanMsg->angle_min   +
scanMsg->angle_increment*i;
   }
}
scan_range_ = smallest_dist;
```
Figure 22: Algorithm to detect pillar location

Topic: /husky_velocity_controller/cmd_vel

The topic /husky_velocity_controller/cmd_vel provides linear velocity and angular velocity from the node /pid_control_node to the node /gazebo to inform the locomotion of the robot in the simulated environment. It has the message type geometry_msgs::Twist.

The section of the RQT graph in Figure 23 shows the relevant nodes and topics for localization. The relevant nodes are /gazebo and /pid_control_node. The topics are /scan and /husky_velocity_controller/cmd_vel.

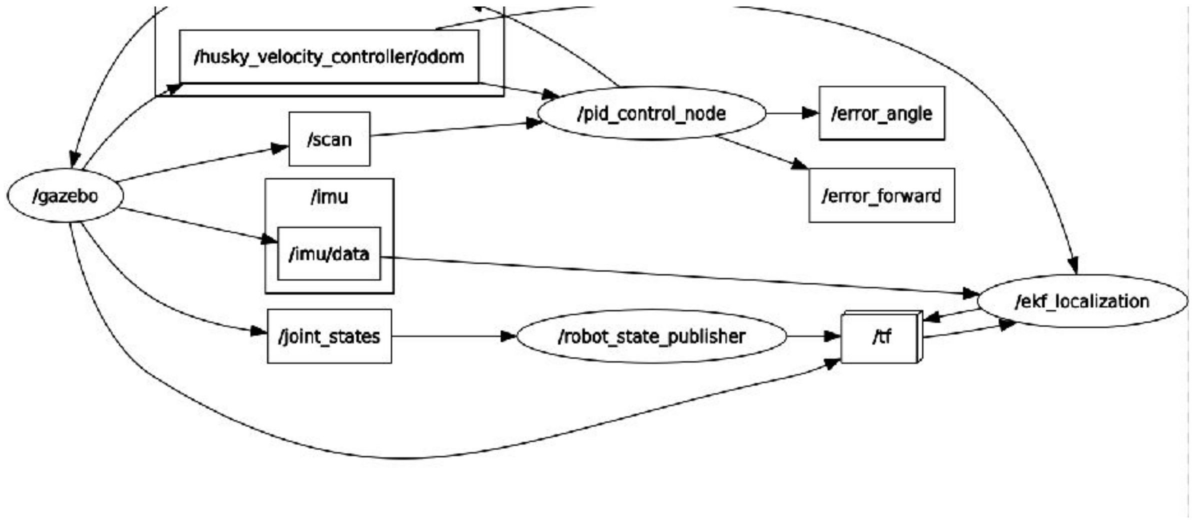


Figure 23: RQT graph of nodes and topics for localization

Node: /gazebo

The node /gazebo is responsible for setting the husky robot and environment for simulation. Based on the forces applied onto the robot during simulation, it provides odometry and 2D LIDAR reading as shown in Figure 21 where the node /gazebo is publishing the odometry through the topic /husky_velocity_controller/odom and the

2D LIDAR through the topic /scan. Only the topic /husky_velocity_controller/odom is used for localization.

Node: /ekf_localization

The node /ekf_localization subscribes to the topic /husky_velocity_controller/odom, /imu/data, and /robot_state_publisher to calculate the new 3D pose and orientation of the different joints in the robot. This enables the simulation to show the robot moving in the environment.

Node: /robot_state_publisher

The node /robot_state_publisher uses the parameters specified in URDF and the joint states from the topic /joint_states to calculate the forward kinematics of the robot and publish the 3D pose and orientation to the topic /tf.[3]

Topic: /husky_velocity_controller/odom

The topic /husky_velocity_controller/odom provide odometry measurements that provide information on pose and twist. The topic is used to calculate the new 3D pose about the robot. The message type is navs_msgs::Odometry.

Topic: /imu/data

The topic /imu/data provide IMU measurements that provide information on orientation, velocity, and acceleration. The topic is used to calculate the new 3D orientation about the robot base frame relative to the world reference frame.[4] The message type is sensor_msgs::Imu.

Topic: /joint_states

The topic /joint_states provide information on the robot joints such as position, velocity, and effort. The message type is sensor_msgs::JointState.

Topic: /tf

[3] http://wiki.ros.org/robot_state_publisher

[4] http://wiki.ros.org/robot_pose_ekf

The topic /tf contains information of the 3D pose and orientation of the robot links.[5]

The message type is tf::tfMessage.

[5] http://wiki.ros.org/robot_state_publisher