TurtleSim Task Report

Setup:

- Install ROS noetic on Ubuntu 20.04 from here. Install full-desktop install which automatically installs turtlesim.
- Clone this repo to catkin workspace and build it. Source the workspace after building.

Note: Might need to make python files executable. To do that use : `chmod +x /path_to_file`

💡 The git repo containing the src folder on catkin workspace is found here.

Goals

Goal 1: Control Turtle

The aim of this goal is to move the turtle from random spawning point to a goal by controlling the linear and angular velocity. Since the turtlebot is a 2D bot, only linear velocity in x and angular velocity in z needs to be calculated. The rest of velocities are made zero.

The linear velocity is calculated using Euclidean distance, while the angular velocity is calculated using steering angle which is the slope of the line joining 2 consecutive points.

Considering q_x and q_y to be current pose of the turtle bot and q_x^g and q_y^g to be the goal position, the Euclidean Distance d is calculated as:

$$d = \sqrt{(q_x - q_x^g)^2 + (q_y - q_y^g)^2}$$

The steering angle is calculated as

$$tan^{-1}(\frac{q_y^g - q_y}{q_x^g - q_x})$$

Proportional Control for linear velocity is a gain value multiplied by linear velocity calculated from Euclidean Distance. For angular velocity it is gain value multiplied by steering angle - yaw of the turtle bot(θ).

This can be verified by launching:

```
roslaunch turtlesim_task goal_1.launch x_val:=2 y_val:=3
```

Here x_val and y_val specify the goal position. If no value is given, the default is set to (1,1). The video result is found here.

PID Controller:

In this implementation, the proportional term is same as the previous implementation. The derivative term is calculated as gain_d*(prev_value - current). The integral term is calculated using gain_i*(current+prev_value), so that the errors add up(integrate) overtime.

This can be verified by launching:

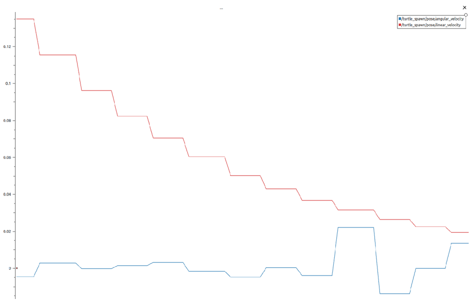
```
roslaunch turtlesim_task goal_1_PID.launch x_val:=2 y_val:=7
```

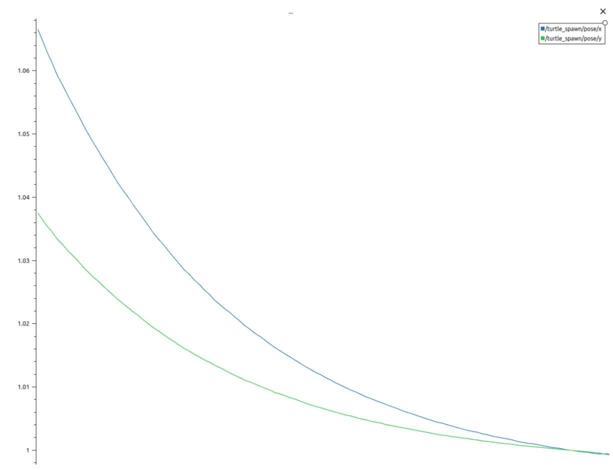
Here x_val and y_val specify the goal position. If no value is given, the default is set to (1,1). The video result is found here.

Optional Task:

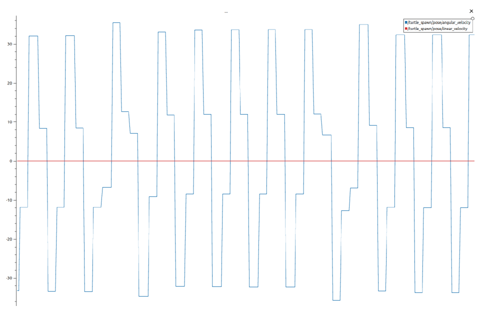
Effect of various gain parameters by observing the state values of x and y along with inputs linear and angular velocities.

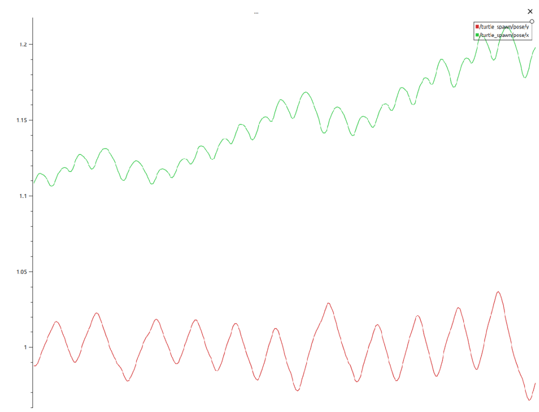
1) Tuned PID: As seen, the input converge to zero and hence the states x and y reach the specified goal of (1,1)



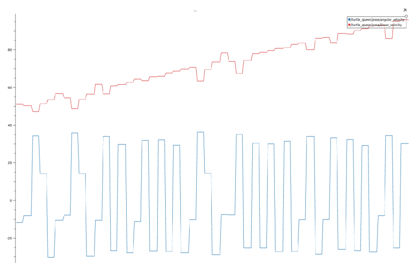


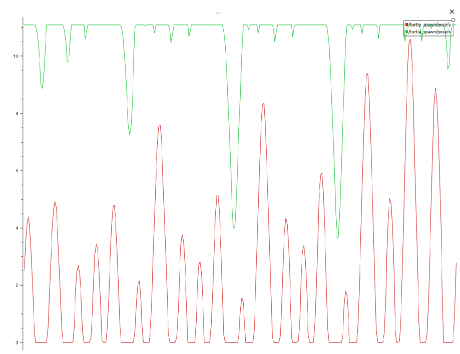
2) High Kd and Ki: As the differential gain and integral gain is high, the states oscillate a bit before converging.





3) High Ki: Without Kd to compensate, there is significant amount of undershoot and overshoot.





Goal 2 : Rotate turtle in circle

The goal is to make a turtle move in a circle given velocity(v) and radius(r). Since the velocity is already known, the linear velocity is set to v. The velocity v can be written as $v = r\omega$ where ω is the angular velocity. Therefore angular velocity is set as $\frac{v}{\omega}$.

After making a turtle \turtle_spawn is made to move in a circle, another instance of turtle /turtle_PT is spawned that follows /turtle_spawn using the topic /rt_real_pose published every 5 sec. The /turtle_PT is the same as the PID from previous goal.

This can be verified by launching:

```
roslaunch turtlesim_task goal_2.launch radius:=3 velocity:=3
```

Here radius and velocity variables are to set desired radius and velocity for the /turtle_spawn to follow. By default they are set to radius = 3 and velocity = 2. The video for the same is found here.

As seen from the video, one the /turtle_PT reaches /turtle_spawn, it will remain on the circle.

Goal 3 : Chase turtle

A turtle named /turtle_RT is spawned to make it move it a circle with the default radius 3 and default velocity 2 as seen in the previous goal. After 10 sec, another instance named /turtle_PT is spawned randomly that chases after /turtle_RT. Since the radius is set at 3units, the /turtle_PT stops the chase when it is at 0.3 units distance. The /turtle_PT gets pose from the topic /rt_real_pose.

To set acceleration limits on \turtle_PT:

Since turtlesim does not have provision to set acceleration, this has been done through velocity. When the acceleration calculated as $\frac{dv}{dt}$ exceeds acceleration limit, the acceleration is set to maximum and velocity final v_f is calculated from the equation below. Here dt is set to 1/10 because the communication rate is set to 10Hz. (P.S. This is the first time I am imposing acceleration limits via velocity. Thank you for considering.)

$$a_{max} = a = \frac{dv}{dt} = \frac{v_f - v_i}{d_f - d_i}$$

This can be verified by launching:

```
roslaunch turtlesim_task goal_3.launch
```

By default they are set to radius = 3 and velocity = 2. The video for the same is found here.

As seen from the video, where the /turtle_PT reaches within 0.3 units of /turtle_RT, the chase is stopped. It can also be observed from the velocity plot in the video, because of the acceleration limits, while deceleration, we can see that the plot is clipped in certain areas.

Goal 4 : Escape turtle (Optional)

A turtle named /turtle_RT is spawned that moves randomly. Another instance /turtle_PT is spawned that can move half as fast as /turtle_PT. If the velocity of PT is more that RT/2, the velocity is set to RT/2. PT has access to the velocity of RT as well as /rt_real_pose.

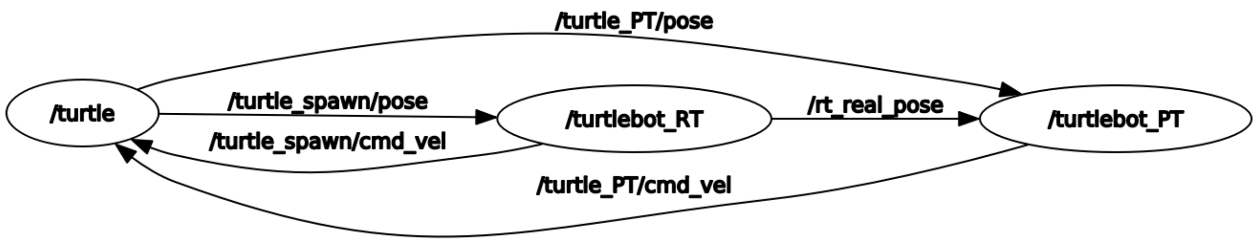
This can be verified by launching:

```
roslaunch turtlesim_task goal_4.launch
```

The video for the same is found here. As seen from the video, where the /turtle_PT reaches within 0.3 units of /turtle_RT, the chase is stopped.\

Explanation of communication stack

Since goal_3 is the most complicated tasks, the communication stack is only explained for goal_3 in the interest of length of the report and rest are left out for brevity.



The pose topics of both RT(/turtle_spawn/pose) and PT(/turtle_PT/pose) are communicated to both the turtles receptively by the /turtle simulator node. The nodes /turtlebot_RT and /turtlebot_PT calculate velocity based on the pose received and the velocity is communicated back to /turtle simulator node to move the turtlebots. The node /turtlebot_PT also receives the position of /turtlebot_RT via the topic /rt_real_pose every 5 seconds. The node /turtlebot_PT uses the pose received via the topic /rt_real_pose as the goal pose.

Things to improve given more time

- Improve the code readability. There are too many functions and variables that were constantly repeated across
- Explain the impact of different gain values in PID
- Write better logic for follower turtle in Goal 3 and 4.
- Work on taking the pose information from the RT(if allowed) and write a better escape logic. Also write a better logic for path of RT, possibly using sub-sampling or even MPC.