Team 7: Race Control

Shone Patil, Takashi Yabuta, Sahil Shah

Abstract - This paper represents the integration of micro-controllers which input odometry based path data as well as a localization package to autonomously follow a path. The two controllers we will be focusing on include the PID and LQR controllers. The original goal was to integrate with the local path team to autonomously navigate on a real robot, but for proof of concept, we established functionality using simulators like F1TENTH.

Key Words- Controller, PID, LQR, path, localization, cross-track error

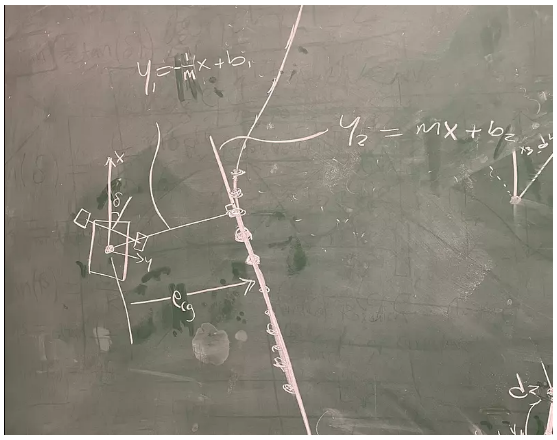
1. INTRODUCTION

A control system manages, commands, directs, or regulates the behavior of other devices or systems using control loops. It can range from a single home heating controller using a thermostat controlling a domestic boiler to large industrial control systems which are used for controlling processes or machines. Micro-controllers are the crux and backbone of many of these new control-feedback systems. In order to regulate something simple like a miniature robot-car's control feedback system which can autonomously follow a path line, we will be utilizing the PID and LQR controllers, which both continuously calculate error values and metrics to regulate the control system and input resulting commands. In the case of a car following a line, we will be inputting cross-track error and heading error values into the systems to regulate the direction the car moves relative to the path line assigned.

2. ERROR METRICS

2.1 Cross-Track Error

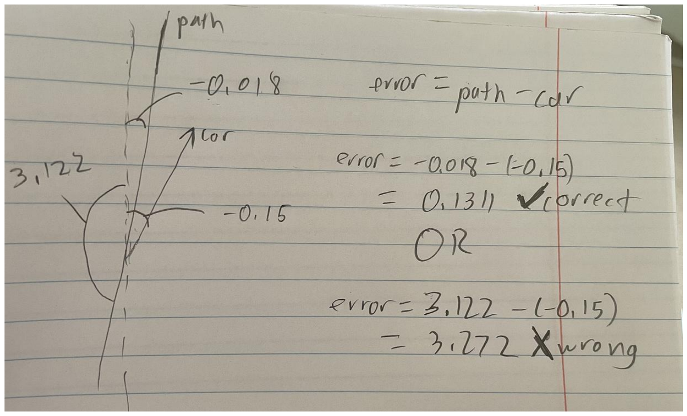
Cross-Track Error is the offset of how far the car is off from the track perpendicularly. This error informs whether the car should turn towards or away from the track. We coded this in our package and accounted for different edge cases where the two closest points to the car were the same or created zero division errors. The main usefulness of this error is to tell us how far from the track the car was and the best way to adjust. Our controllers took this info and based on the distance, adjusted both the steering angle and throttle values that were published to the car. We made sure the subscription rate was fast enough such that the car could quickly calculate cross-track error in order to adjust the car's movements in as close to real time as possible.



Dominic's Explanation of Cross-Track Error

2.2 Heading Error

Heading error is how far off the orientation of the car is from the track's orientation. This error helps with aligning the car's direction with the track, especially when the cross-track error is 0. The reason for having this error is to have a better idea of how aggressively the car needs to turn in order to readjust to the track. Using only cross-track error runs the risk of not being able to account for if the car is already facing the raceline or not and whether it should turn in those cases. Additionally, heading error helps with giving the controller more accurate information to minimize through the throttle and steering angle quantities. In our case, our PID controller used just the cross-track error while the LQR controller used both the cross-track error and heading error in order to minimize offset and correct the car's movements. While we workshopped a version of the PID controller that also used heading error, we decided to not go with it in the end since we ran into some time constraints. In the future, we think it is still worth exploring since PID controllers are much easier to carry out parameter tuning in comparison to the more complex and heavily parameterized LQR controller.



Error Offset Modified to Account for Direction

3. CONTROLLERS

3.1 PID

PID, or Proportional-Integral-Derivative, which computes a control action using tracking error from those three actions. It was easier to implement and served as our baseline to compare our more sophisticated controllers to. It uses a control loop feedback mechanism to control process variables and is an accurate and stable controller. We optimized using just the proportional portion in order to get the car to work on a baseline level. After obtaining a good starting performance of the car moving back and forth across the path, we added in the integral and derivative portions to smooth out the movement. This helped greatly in adjusting the car's aggressiveness when it came to both turns and straightaways. In the F1TENTH simulator, we optimized this controller just to get an idea of what the process would be to tune the parameters such that we could do the same on the real robot in the ideal case.

3.2 LQR

LQR, or Linear Quadratic Regulator which is used for efficient control over linear systems utilizing control costs and quadratic stateshe benefits to this method come in its lower cost to compute and customizable parameters Q and R

4. CHALLENGES FACED

There were many issues and challenges faced which took weeks to resolve. Initially, after utilizing Dominic's code for the PID controller, we ran the code on the simulator, and the car would randomly veer off and not follow the line. We consulted the TA's as to what could be the problems. After trying multiple times, we decided to go back to square one and output the odometry data which was used to create the race path in the simulation. The odometry data showed the first 6000 rows as 0 values, meaning that the car was initially at rest. Once we deleted those duplicate values, the car stopped veering in the beginning. We also found that the code for the PID controller did not implement the car's orientation, meaning that it did not take into account which direction the car was facing relative to the right or the left of the path. We were stumped as to how to have the car update its error values and how the program would tell the car to steer if it didn't know which side of the path it would be on. Regardless, we found that the PID didn't need to utilize orientation since the car slope relative to the path was incorrectly calculated and the issue was with the signs. We got the PID implementation to work and we moved onto the LQR implementation, where we faced many issues. One issue that we faced was finding out why our cross track error was outputting incorrect values. It took a while to debug but it was due to a delay before the program subscribed to the path. We increased the buffer size and limited the delay, which resulted in more accurate CTE values. Moreover, when initially testing the LQR controller, the car in the simulation wasn't turning properly and it resulted in the car veering in random directions. We found after multiple hours that the issue was with the CTE calculation incorrectly partitioning the cross track-error calculation in the code. Initially the code was partitioning the closest distances in the list of error distances and swapping them randomly, which resulted in random theta values being inputted to the program, which told the car to veer at an incorrect angle. The car would initially be following the path, and then the incorrect input would be placed, resulting in the car veering off. Once we eliminated this random partition, the car performed much better. We also found multiple typos and bugs in both the initial baseline PID and LQR code, which could explain the random errors.

5. CONCLUSION

At the beginning of this project, the ideal goal was to input a localized pose from Team F and path from Team E and implement a controller with the real robot to follow the path as best as possible. In order to do this, we would implement three controllers, PID(baseline), LQR, and MPC-which we didn't end up implementing since we were running out of time. First, we tested the PID and LQR controller implementations using the F1TENTH Simulator, Rvis2, and the prebuilt code provided to use and modified to our needs. We created the virtual track by driving the real robot around and collected the odometry data values. Then PID and LQR tuning ensued, with many errors and debugging to do. The implementations utilized cross-track error and heading error to iteratively follow the path based on the closest distance values to the path in relation to the car and adjust its steering based on distance values. LQR tuning turned out to be trickier than it should have been due to multiple errors in the error calculations and if given more time, we could have optimized the process for both the robot and simulation. Since we ran out of time, we were not able to implement the MPC controller, integrate with the local path team to run in the SVL simulator, and run both controllers on the real robot. In terms of the applications of the microcontroller, we learned about its importance in the nature of machines and robotic systems. By reducing the size and cost of a typical microprocessor system, microcontrollers make it economical to digitally control more devices and processes as well as scale large robotic control systems, especially those in factories and industry.

6. DEVELOPMENT TIMELINE

Week 7:

- Created demo path by dumping ROS2 bag of odometry info after teleoping car along path in RVIZ simulation.
- Set up a team repository to manage all our controllers (PID, LQR, MPC - if time permits)
- Started implementing basic PID controller in simulation

Week 8:

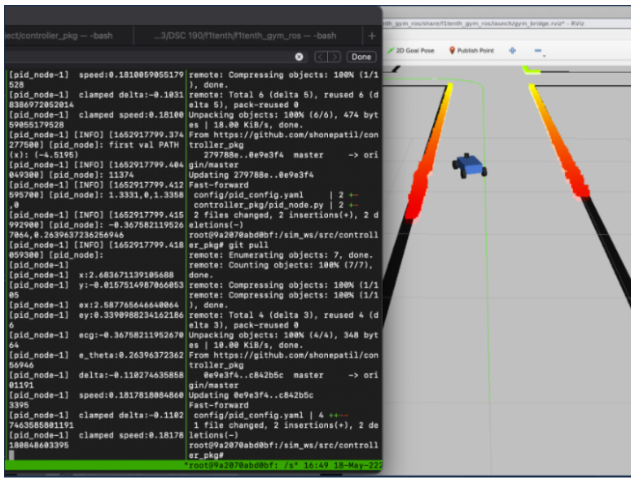
- Trying to make the vehicle follow the path using cross-track error calculation.
- Fixed issue where there was delay before subscribing to path, which resulted in incorrect cross-track error values.
- Using different methods to calibrate error calculation and debug.

Week 9:

- Got the PID controller to work, car follows path around track, error was with car slope calculation.
- Implementing a basic LQR controller in simulator.
- Fixed issue where partitioning of cross track-error calculation was resulting in closest distances of points in list being swapped, resulting in car turning in different direction.
- Made progress with the car following the path.

Week 10:

- Verified LQR controller to work in simulation.
- Explored different Localization packages.
- Sync-up/collaborate with Local Path planning team to publish and subscribe data in correct form.
- Sync-up with Global Path team to specify path as coordinates


ject/controller_pkg — -bash
...3/DSC 190/f1tenth/f1tenth_gym_ros — -bash
Done
2D Goal Pose
Publish Point

```
[pid_node-1]  speed:0.1810059055179
528
[pid_node-1]  clamped delta:-0.1031
8386972052014
[pid_node-1]  clamped speed:0.18100
59055179528
[pid_node-1] [INFO] [1652917799.374
277500] [pid_node]: first val PATH
(x): (-4.5195)
[pid_node-1] [INFO] [1652917799.404
049300] [pid_node]: 11374
[pid_node-1] [INFO] [1652917799.412
595700] [pid_node]: 1.3331,0,1.3358
,0
[pid_node-1] [INFO] [1652917799.415
992900] [pid_node]: -0.367582119526
7064,0.2639637236256946
[pid_node-1] [INFO] [1652917799.418
059300] [pid_node]:
[pid_node-1]
[pid_node-1]  x:2.683671139105688
[pid_node-1]  y:-0.0157514987066053
05
[pid_node-1]  ex:2.587765646640064
[pid_node-1]  ey:0.3390988234162186
6
[pid_node-1]  ecg:-0.36758211952670
64
[pid_node-1]  e_theta:0.26396372362
56946
[pid_node-1]  delta:-0.110274635858
01191
[pid_node-1]  speed:0.1817818084860
3395
[pid_node-1]  clamped delta:-0.1102
7463585801191
[pid_node-1]  clamped speed:0.18178
180848603395
```

```
remote: Compressing objects: 100% (1/1
), done.
remote: Total 6 (delta 5), reused 6 (d
elta 5), pack-reused 0
Unpacking objects: 100% (6/6), 474 byt
es | 18.00 KiB/s, done.
From https://github.com/shonepatil/con
troller_pkg
   279788e..0e9e3f4  master     -> ori
gin/master
Updating 279788e..0e9e3f4
Fast-forward
 config/pid_config.yaml     | 2 +-
 controller_pkg/pid_node.py | 2 +-
 2 files changed, 2 insertions(+), 2 d
eletions(-)
root@9a2070abd0bf:/sim_ws/src/controll
er_pkg# git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7),
done.
remote: Compressing objects: 100% (1/1
remote: Compressing objects: 100% (1/1
), done.
remote: Total 4 (delta 3), reused 4 (d
elta 3), pack-reused 0
Unpacking objects: 100% (4/4), 348 byt
es | 10.00 KiB/s, done.
From https://github.com/shonepatil/con
troller_pkg
   0e9e3f4..c842b5c  master     -> ori
gin/master
Updating 0e9e3f4..c842b5c
Fast-forward
 config/pid_config.yaml | 4 ++--
 1 file changed, 2 insertions(+), 2 de
letions(-)
root@9a2070abd0bf:/sim_ws/src/controll
er_pkg#
"root@9a2070abd0bf: /s" 16:49 18-May-22
```

CITATIONS

[1] https://www.autonomousrobotslab.com/pid-control.html
We referenced this resource for information regarding the PID controller and how it was implemented as well as its function.

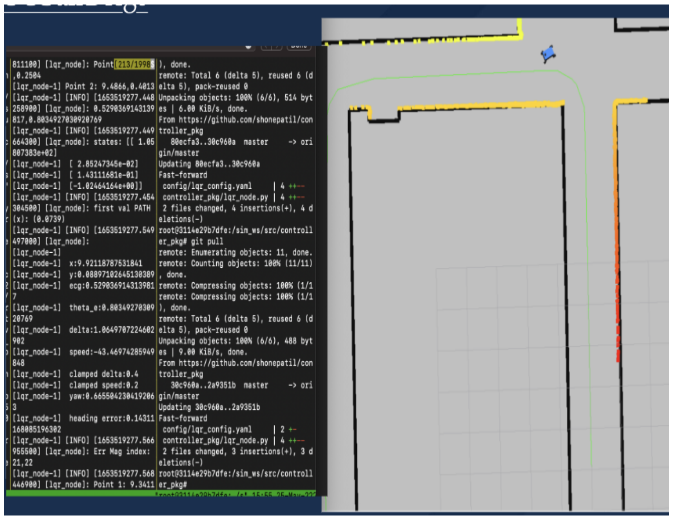
[2]https://arxiv.org/pdf/2009.13175.pdf#:~:text=LQR%20focuses%20on%20non%2Dlinear,actual%20system%20requires%20its%20linearization
We referenced this resource for information regarding the LQR controller and how it is implemented as well as its function.

[3] https://www.youtube.com/watch?v=E_RDCFOlJx4
We referenced this youtube video for information regarding the LQR controller and how it worked.

[4](https://www.youtube.com/watch?v=wkfEZmsQqiA&ab_channel=MATLAB)
We referenced this youtube video for information regarding the PID controller and how it worked.

[5]https://github.com/shonepatil/controller_pkg/tree/master/controller_pkg
This was initially the LQR and PID controller packages which we workshopped and modified from Dominic's code.

[6]https://github.com/shonepatil/path_pkg
This is what we utilized for mapping the paths and experimenting with different paths.



```
811100] [lqr_node]: Point[213/1998
,0.2504
[lqr_node-1] Point 2: 9.4866,0.4013
[lqr_node-1] [INFO] [1653519277.448
258900] [lqr_node]: 0.5290369143139
817,0.8034927030920769
[lqr_node-1] [INFO] [1653519277.449
664300] [lqr_node]: states: [[ 1.05
807383e+02]
[lqr_node-1]  [ 2.85247345e-02]
[lqr_node-1]  [ 1.43111681e-01]
[lqr_node-1]  [-1.02464164e+00]]
[lqr_node-1] [INFO] [1653519277.454
304500] [lqr_node]: first val PATH
(x): (0.0739)
[lqr_node-1] [INFO] [1653519277.549
497000] [lqr_node]:
[lqr_node-1]
[lqr_node-1]  x:9.92118787531841
[lqr_node-1]  y:0.08897102645130389
[lqr_node-1]  ecg:0.529036914313981
7
[lqr_node-1]  theta_e:0.80349270309
20769
[lqr_node-1]  delta:1.0649707224602
902
[lqr_node-1]  speed:-43.46974285949
848
[lqr_node-1]  clamped delta:0.4
[lqr_node-1]  clamped speed:0.2
[lqr_node-1]  yaw:0.665504230419206
3
[lqr_node-1]  heading error:0.14311
168085196302
[lqr_node-1] [INFO] [1653519277.566
955500] [lqr_node]: Err Mag index:
21,22
[lqr_node-1] [INFO] [1653519277.568
446900] [lqr_node]: Point 1: 9.3411
```

```
), done.
remote: Total 6 (delta 5), reused 6 (d
elta 5), pack-reused 0
Unpacking objects: 100% (6/6), 514 byt
es | 6.00 KiB/s, done.
From https://github.com/shonepatil/con
troller_pkg
   80ecfa3..30c960a  master     -> ori
gin/master
Updating 80ecfa3..30c960a
Fast-forward
 config/lqr_config.yaml     | 4 ++--
 controller_pkg/lqr_node.py | 4 ++--
 2 files changed, 4 insertions(+), 4 d
eletions(-)
root@3114e29b7dfe:/sim_ws/src/controll
er_pkg# git pull
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11)
, done.
remote: Compressing objects: 100% (1/1
remote: Compressing objects: 100% (1/1
), done.
remote: Total 6 (delta 5), reused 6 (d
elta 5), pack-reused 0
Unpacking objects: 100% (6/6), 488 byt
es | 9.00 KiB/s, done.
From https://github.com/shonepatil/con
troller_pkg
   30c960a..2a9351b  master     -> ori
gin/master
Updating 30c960a..2a9351b
Fast-forward
 config/lqr_config.yaml     | 2 +-
 controller_pkg/lqr_node.py | 4 ++--
 2 files changed, 3 insertions(+), 3 d
eletions(-)
root@3114e29b7dfe:/sim_ws/src/controll
er_pkg#
```