paper: Implementation of the Adaptive Pure Pursuit Controller

CD-Media: Papers
programming
autonomous
motionprofiling

edf42001 #1 January 2, 2019, 6:11am

Thread created automatically to discuss a document in CD-Media.

Implementation of the Adaptive Pure Pursuit Controller
by: **edf42001**

A guide to implementing the pure pursuit path following algorithm on an FRC robot.

Team 1712 had their most succesful season this year, due in part to implementing the pure pursuit algorithm in our autonomous. It is fast, reliable, accurate, and makes it easy to create and adjust paths on the fly. At its heart, the pure pursuit algorithm consists of driving towards a point on the path in front of you, which is a very intuitive way to follow a path. We include a discussion on generating smooth velocity setpoints for the robot to follow, even though it is not strictly part of the pure pursuit algorithm.

Our implementation is programmed in LabVIEW, and our code is here: https://github.com/Dawgma-1712/new-FRC-2018. The pure pursuit VIs are in Robot-Project/Drive/PurePursuit. To see the algorithm in action, open and run TestPurePursuit. The implementation can be found in the PurePursuit VI.

If you have any questions or discover any issues, please feel free to reply to this thread or email us at frc1712@gmail.com.

Pure Pursuit Implementation.pdf (674 KB)

Limelight 2019.5 - 3D, Breaking Changes, and Darker Images

Help With Pure Pursuit Implementation

AriMB #2 January 1, 2019, 6:30pm

Very interesting algorithm, and the paper was written very understandably. Definitely something I'll be sharing with my programming team. Kudos!

A few questions:
• Why did you take away the initial acceleration for the target velocity calculations and then add in the rate limiter to compensate instead of just starting with a small positive target velocity (so the

robot will start to move) and limiting the acceleration via the target velocity motion profile?
• How well does this method handle 180° turns? Can it drive backwards or only forwards?
• Did you play around with the difference between increasing path smoothness (from 2168's algorithm) and increasing lookahead distance? It seems like the two would do the same thing, possibly to the point where smoothing the path might not be necessary.
• Did you ever see the robot get far enough off the path that it couldn't recover (i.e. further than the lookahead distance away). If so, what did you have the robot do?
• How much of these calculations was done pre-match and what was done in "real time"?

---

edf42001 #3 January 1, 2019, 6:30pm

Thanks for reading! Fair warning, I enjoy delving into a lot of detail, so here goes!

> Why did you take away the initial acceleration for the target velocity calculations and then add in the rate limiter to compensate instead of just starting with a small positive target velocity (so the robot will start to move) and limiting the acceleration via the target velocity motion profile?

Because there is no interpolation between points when finding the target velocity, it jumps between values, especially at low speeds. Removing the acceleration from the velocity profile and adding in the rate limiter makes the robot follow a more smooth acceleration curve at the beginning of the path.

Otherwise, both methods should work, you just need to make another pass through the target velocities in the forward direction to apply the acceleration limits. One possible disadvantage of using a rate limiter is that if your robot is traveling faster then it should, it will drive through deceleration points faster then the rate limiter allows it to decelerate and could overshoot the end of the path. This hasn't been an issue for us.

> How well does this method handle 180° turns? Can it drive backwards or only forwards?

We have never tried a path that goes straight forwards then straight backwards, but we did do a small U-turn once in testing, which worked pretty well because the robot slows down around sharp turns. If the turn is too small the robot could turn too early and not slow down because the closest point would never be the one with the sharp curvature and slow target velocity. This is where it could be helpful to adjust the lookahead distance based on robot speed or path curvature, so it follows the sharp turn more precisely. I suggest experimenting with the simulator, which I would love to do to help answer your question but I have to go on this thing called "vacation". Sigh.

Amazingly, to make the robot drive backwards all you have to do is negate the target velocity before sending it to the left/right wheel speed calculations. We haven't tried this on the real robot yet, only in the simulator.

> Did you play around with the difference between increasing path smoothness (from 2168's algorithm) and increasing lookahead distance? It seems like the two would do the same thing, possibly to the point where smoothing the path might not be necessary.

We haven't really played with this, but here's why I think smooth paths are better:

1. When you increase the lookahead distance to try to follow an unsmoothed path smoothly, the robot cuts corners, making it difficult to make small, precise turns.
2. Smoothing the path and using a smaller lookahead distance to precisely follow it means that when you graph the path you see precisely where the robot will be driving ahead of time.

So for best results, we smooth the path, use a smaller lookahead distance so we can go around sharp turns, and if we want sharp and less sharp turns in the same path we truncate the corner of the less sharp turn (so one 90 angle turns into two 45s).

> Did you ever see the robot get far enough off the path that it couldn't recover (i.e. further than the lookahead distance away). If so, what did you have the robot do?

While this has never happened to us, if the robot can't find a lookahead point it will use the value of the last one and steer back onto the path. However, it might be better instead to use a point a bit farther down the path than the closest point, because:

1. This point will travel with the robot as the robot continues forward, unlike the last lookahead point, so the robot won't have to do a u-turn to get back to the point.
2. There might be a reason for starting the robot off of the path, in which case there is no last lookahead point.
3. Another place the robot can't find the lookahead point is at the end of the path, and using a point a few ahead of the closest point will make the robot aim for the end of the path, like it should.

> How much of these calculations was done pre-match and what was done in "real time"?

The smoothed paths are calculated pre-match and loaded into files, because for large smoothing amounts the smoothing algorithm can take a decent amount of time to work. Once autonomous starts, the path data is read and the path curvature and velocity setpoints are calculated. Then odometry, finding the lookahead point, curvature, closest point, left/right wheel speeds, etc., is done in real time.

AriMB #4 January 1, 2019, 6:32pm

I've been meaning to get into autonomous path-following algorithms, and this white paper seemed like an intuitive place to start. Using the steps as explained here with a few minor changes, I rewrote the code from scratch in Python (here). The white paper was very easy to follow, and the whole thing came together pretty smoothly. I could definitely see my team using something like this in the future for autonomous path-following.

Thank you again for this resource!

Dan_Katzuv #5 January 1, 2019, 6:40pm

What are the meanings of “adaptive” and “pure” in this algorithm?

edf42001 #6 January 3, 2019, 6:49am

> **Dan Katzuv:**
>
> What are the meanings of “adaptive” and “pure” in this algorithm?

Great question! I believe “pure” refers to how the path following algorithm is based purely on pursuing the lookahead point. “Adaptive” refers to modifications of the basic pure pursuit algorithm. For example, since larger lookahead distances are more stable, one adaption could be to scale the lookahead distance proportionally with tracking error, so that if your robot accidentally gets far off the path it won’t break itself while vigorously trying to swerve back onto the path.

I’ll admit I just borrowed the name from Team 254’s code, who may have gotten it from this paper:
https://www.ri.cmu.edu/pub_files/pub1/kelly_alonzo_1994_4/kelly_alonzo_1994_4.pdf
So thanks for making me think about it! 😃

1 Like

Galen #7 January 1, 2019, 6:45pm

Hello ! I have a question and I need an answer as soon as possible, please!

How did you run your VI’s in parallel in a myRIO, or in your case in a robotRIO ?
If you didn’t do that, what method did you use to acquire and process data in real time ?

AriMB #8 January 3, 2019, 6:50am

> **Galen:**
>
> Hello ! I have a question and I need an answer as soon as possible, please!
>
> How did you run your VI’s in parallel in a myRIO, or in your case in a robotRIO ?
> If you didn’t do that, what method did you use to acquire and process data in real time ?

In LabVIEW you can run two loops in parallel by putting them both in the main VI without any wires connecting them.

This is not really the place to ask these questions, since this thread is for discussion of the Pure Pursuit implementation. If you have any further questions you can PM me, or post your own thread. You should know though that this forum is specifically for the FIRST robotics competition, so if you are not affiliated with the program you may finding people are less inclined to answer your questions.

Anon10W1z #9 January 28, 2019, 6:33pm

We've gone ahead and implemented this algorithm in Java for anyone interested. More details are at the wiki.

1 Like

Dan_Katzuv #10 February 25, 2019, 7:52pm

We used your great paper and implemented the code in java. Many thanks!
However we have an issue with the end, we reach an error of about 1-2" in the endpoint. Did you get a higher accuracy?

edf42001 #11 February 26, 2019, 1:28am

I don't know. Last year accuracy wasn't really required as the scoring area was huge, and we haven't tested it on this year's robot yet. But in our testing last year it did appear very consistent, so if our robot ended up in an inaccurate, but consistent, place we simply adjusted the path until it ended up where we wanted it to. We also noticed that sharp turns at the end of the path lead to inaccurate results, so try to remove those if you can. Lowering the lookahead distance can also help the robot follow a path more accurately.

Yeah, pure pursuit isn't known to excel at stopping in the right place, and this problem is increased this year because the robot has to drive of the HAB platform. Let me know if you're able to get more accurate results!

Jason2073 #12 June 6, 2019, 4:34pm

Hello, I realize this thread is quite old, but I still find it incredibly useful, and I was wondering how to calculate the center point of the circle that connects the current robot position to the lookaead point. Any help is much appreciated, and thank you for all you have done already!

edf42001 #13 June 6, 2019, 6:15pm

No problem. To find the center of that circle, first find its radius. Radius is equal to 1/curvature, and since the curvature is signed to indicate whether the circle is to the left or the right of the robot, so will be the radius. Next, due to the way the controller is designed, the circle is tangent to the way the robot is facing. So the center is 1 radius away in the direction perpendicular to the robot.

To find this point you can take the unit vector in the direction the robot is facing, rotate it by 90 degrees, multiply it by the radius, and finally add it to the current robot's location:

```
radius = 1/curvature
theta = //current robot angle
Vector toCenter = Vector(cos(theta-90), sin(theta-90)) //unit vector perpend
toCenter *= radius //scale by radius of circle
center = toCenter + currentRobotLocation //add that vector to the robot's cu
```

I may have messed up the signs, so if you find your circle is in the wrong direction just invert the toCenter vector.

Just wondering, are you looking for the center of this circle so you can make a visualization?

Jason2073 #14 June 6, 2019, 6:18pm

Ah that makes sense, i missed that the circle is tangent to the robot’s heading so it was becoming much harder than I expected. And yes im doing this for visualization, where im just plotting data to a csv file and then graphing with GNUplot.