15375 Control Award Submission

For this years autonomous period, 15375 employed a variety of strategies to ensure consistency and reliability, and also optimize performance. As a rookie team, we first needed to get the basics down, which meant using motor encoders and the IMU to its maximum capacity.

Motor Encoders:

We used the motor encoders on our drive train wheels to track our position and go set distances. Specifically, we employed them for odometry and localization, so we could have an absolute location of where our robot was on the field (More on this will be discussed later). Motor encoders provide PID velocity control to ensure that the motor velocity stays constant and doesn't oscillate.

IMU:

We used the imu to perform accurate turns using PID and keep track of our robots heading at any point in time. The imu was critical in ensuring our robot was oriented correctly.

PID Control:

When travelling to a setpoint, whether that be a target angle or position, travelling at a constant velocity will guarantee overshoot. In order to compensate for this, we would like to slow down as we reach the setpoint. The ideal why to do this is to use Proportional, Integral, and Derivative control which correct for error at a specific time, steady state error, and future error respectively. In our code, we tuned coefficients for turns (heading), axial movements, and lateral movements. This ensured a small margin of error for all movements and turns during autonomous.

We follow the formula

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

To calculate the desired output based on feedback error

Feedforward control.

In order to ensure further accuracy however, we applied feedforward control that is fed into the controller before the desired output is performed. Specifically, we use a constant k_v = max voltage / max velo. This reduces the amount a steady state error, allowing for faster tuning times and eliminates the need for an integral term (which has negative side effects like windup)

When implementing feedforward in addition to feedback, we can do the following:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \left(\frac{de(t)}{dt} - v \right) + k_v v + k_a a$$

Trajectory Following

For this years autonomous period, we opted to use motion profiling to increase our effeciency and learn a new skill. Specifically, we interpolated quintic splines for our robot to follow, rather than wasting time following linear movements and seperate turns.

Before we get in to the interpolation and generation of our profile, we need some sort of odometry to know our robots location relative to the field coordinates.

For holonomic drives, we can employ the following kinematic formulas:

Mecanum Kinematics

Each wheel has an additional x and y velocity component due to the vehicles rotational velocity ω_v given by

$$V_x r = -Y_n \cdot \omega_v$$

$$V_y r = X_n \cdot \omega_v$$

The total linear velocity given by the vector componenets is

$$V_{x_n} = V_x + V_{x_{rn}} = V_x - Y_n \cdot \omega v$$

$$V_{y_n} = V_y + V_{y_{rn}} = V_y - X_n \cdot \omega v$$

$\frac{V_x + V_y \cdot \tan\theta}{r}$ = wheel velocity

Odometrical Calculations

We can calculate axial, lateral, and angular velocities from the wheel velocities, and then obtain a velocity vector as follows:

We can calculate the robots pose velocity like follows $$v(t) = (v_x \cos (\theta_0 + \omega\tau) - v_y \sin(\theta_0 + \omega\tau), v_x \sin (\theta_0 + \omega\tau) + v_y cos(\theta_0 + \omega\tau)$$

Integrating results in

$$s(t) = \int_0^{\Delta t} v(\tau)\,d\tau = (\frac{v_x}{\omega}[\sin\theta_1 - \theta_0] + \frac{v_y}{\omega}[\cos\theta_1 - \cos\theta_0]), (-\frac{v_x}{\omega}[\cos\theta_1 - \cos\theta] + \frac{v_y}{\omega}[\sin\theta_1 - \sin\theta_0])$$

which is our updated robot pose (x and y component).

We can then use the imu to calculate the heading component.

Quintic Spline Trajectories

Our splines will be expressed as a vector of single variable functions for the x and y component.

Quintic splines consist of a series of segments combined into a single peicewise curve. Each segment is a parametric curve with a quintic polynomial for each component.

For most of our segments in FTC, we will use one segment, which will also allow for some more brevity in our notation.

$$r(t) = \begin{cases} x(t) = a_x t^5 + b_x t^4 + c_x t^3 + d_x t^2 + e_x t + f_x, \\ y(t) = a_y t^5 + b_y t^4 + c_y t^3 + d_y t^2 + e_y t + f_y \end{cases}$$

where $0 \le t \le 1$

Reparametrization to Arc Length

We now need to reparametrize all the segments into a single variable of arclength ($t \in [0,1] \rightarrow s$)

s is simply the displacement along the curve.

thus: $|r'(s)| = 1 = \sqrt{(\frac{dx}{ds})^2 + (\frac{dy}{ds})^2}$as 1 unit

It is simple to represent $s(t)$ as arclength as a function of t

$$s(t) = \int_0^t |r'(\tau)|\, d\tau = \int_0^t \sqrt{\left(\frac{dx}{d\tau}\right)^2 + \left(\frac{dy}{d\tau}\right)^2}\, d\tau$$

We can numerically integrate this and match resulting arc lengths to times

We can now determine $r(s)$ by composition: $r(t(s))$

We can now differentiate to yield the parametrized derivative:

$$r'(s) = r'(t) \cdot t'(s) = \frac{r'(t)}{s'(t)} = \frac{r'(t)}{|r'(t)|}$$

We realize this makes sense as we can realize that $r'(s)$ = 1.

Now let us set $s(t)$ be the state of the motion profile at time t.

We now have

$$v(t) = \frac{d}{dt}[r(s(t))] = r'(s(t))s'(t)$$

Heading

In addition to the translational motion, we also need to interpolate some type of angular motion.

For nonholonomic drive trains, we can update the heading like so:

$\theta(t) = \arctan \frac{y'(t)}{x'(t)}$

For holonomic drive trains, we can consider the heading component to be a completely independent spline.

$$\theta'(t) = \frac{x'(t)y''(t) - x''(t)y'(t)}{x'(t)^2 + y'(t)^2}$$

We can now calculate this derivative at $t = 0$ and $t = 1$

Generating a Profile

We now want to generate a motion profile $s(t)$ = arc length traveled where t is time

Satisfying maximum velocity, acceleration, and angular velocity constraints.

Forward Pass

We look at pairs of consecutive states, where the start state has already had its velocity determined. We now need to find an acceleration that is allowable at both the start and end state

Steps

- Compute the segment constraints (max vel, max acc)
- If the last velocity exceeds the max velocity, we just coast at max velocity
- Else, we can compute the final velocity assuming max acceleration

$$v_f = \sqrt{v^2 - 2a_{max}\Delta s}$$

- If this final velocity is under the maximum velocity, we are good and we can continue accelerating
- Otherwise we want to split the segment:

$$\Delta s_{acc} = \frac{v_{max}^2 - v^2}{2a_{max}}$$

- We can then deaccelerate for Δs_{acc}, and then coast when we reach max velocity
- We now repeate this process starting at the last planning point and going back
- Now we want to obtain our final states, so we now need to compare forward and backward state deltas
- If there is a difference in the displacements of the state, we can split the longer segment into two and add the second segment to our forward state
- Lets define some terms:

  Δs_f , Δs_b are the forward and backward displacements respectively

  case $\Delta s_f > \Delta s_b$:

  We calculate the kinematic state after traveling $\Delta s_f - \Delta s_b$

  case $\Delta s_f < \Delta s_b$:

  We calculate the kinematic state after traveling $\Delta s_b - \Delta s_f$

  we add the resulting kinematic state to its respective state list (forward or backward).

- After we establish forward and backwards consistency, we now need to calculate the end state after the alginement

- We then calculate the minimum velocities at each start and end state... and if the backward state has a lower velocity than the forward state, we calculate the intersection between the two end states and add that to our final states, by using the following equation:

  $$\frac{v_{prev}^2 - v^2}{4a_t a_{prev_t}}$$

- Finally we want to turn the final states into time parametrized motion segments

  v = state velocity

  $$\Delta t = -\frac{v}{a_t} \pm \sqrt{\frac{v^2}{a_t^2} + \frac{2\Delta s}{a_t}}$$

  Note: in order to do this, we need to keep track of the displacements for each state

- We now can generate a motion profile containing in which each state satisfies velocity and acceleration constraints

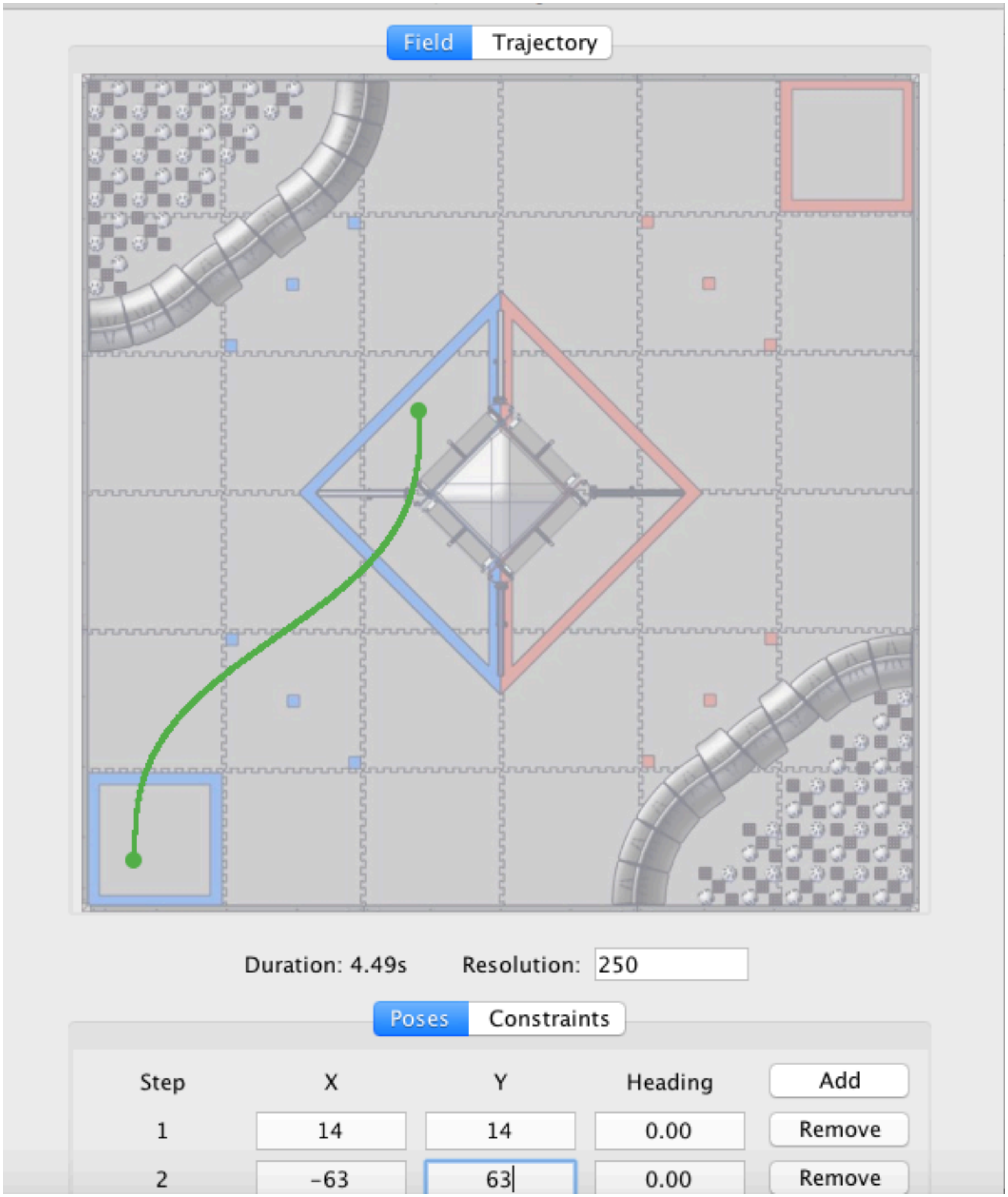
Following the trajectory

In order to follow the trajectory, we need a way to account for error for axial, lateral, and angular motion. To do this, we employ seperate PIDF (PID + Feedforward) controllers for each of these. We calculate the desired velocity acceleration, and heading by taking the first and second derivative of the resulting motion profile. We then compare then feed the errors into each controller to determine the corrected velocity and then map that to wheel velocities, then to motor powers in order to reach the desired state.

Computer Vision

Although much less interesting, we employed OpenCV in order to detect the gold mineral to hit when sampling. Although we considered using the built in tensorflow from the sdk, it proved to be unreliable and too slow. Because opencv is run in native c++ code, we were able to achieve desired and accurate performance, and also the experience of writing a vision algorithm. Hopefully there are more opportunites to use vision next year !!!!

Paths Completed



Duration: 4.49s Resolution: 250

Step	X	Y	Heading
1	14	14	0.00
2	-63	63	0.00

References

- http://www.chiefdelphi.com/media/papers/download/2722
- https://github.com/acmerobotics/road-runner/blob/master/doc/pdf/Quintic_Splines_for_FTC.pdf
- http://www2.informatik.uni-freiburg.de/~lau/students/Sprunk2008.pdf