Software Design

Air Hockey Cable Robot

Last updated by Thomas, 9 March 2026

Overview

3 main software components:

- Computer vision (I (Thomas) don’t know anything about this)
- RL agent
- Motor controller

The purpose of these slides are to specify the interfaces between these components

Goal: destroy everyone at air hockey

Architecture

Btw the motors will each have a PID controller but this is a given

CV pipeline

- Hardware: 2 cameras
- Input: Camera images at 120 fps
- Obtains current position and velocity of puck
  - Can also track the mallet position and velocity to calibrate
  - We can even track the opponent's mallet position and velocity lol
- Output: reports the position/velocity data to the RL agent

RL agent

- Hardware: jetson nano
- Input: position/velocity data from CV pipeline
- Uses mallet/puck position/velocity data to compute mallet movements (and desired shot trajectories)
  - If we want, it is even possible to fake shots or do wall bounces and crazy stuff like that
- Output: sends continuous position/velocity/time data to the motor controller
  - "Where do I need to be, and when do I need to get there?"

Motor controller

- Hardware: Arduino and 4 motor assemblies
- Input: position/velocity commands from RL agent
- Computes the motions required to achieve the position/velocities desired by the RL agent
  - So the RL agent controls the motion at a higher level (and determines the 'critical moments', namely when hitting the puck or receiving a shot), but the motor controller figures out the granular details of the motion in between those points
- Output: voltage/current controls to the 4 motors

CV Pipeline

I (Thomas) don't know anything about the granular details of how CV works, but we have 2 cameras and we should be able to use them to **triangulate the puck and mallet(s)** with decent precision

Questions:

1. Can we design a CV pipeline that is robust to possible angular misalignment?
    a. In other words, as long as the two cameras can "see" the entire playing field, is that good enough for us (even if the cameras are set up at angles that are not precise)?
2. On the coding side, how can we optimize the computation so that the **delay** between image capture and velocity computation is **as small as possible**?

Some basic research from my end suggests that we can use camera calibration techniques and Kalman filters to address both points, but you guys know better than me.

RL Agent

It would be really cool if the RL agent could learn some crazy strats, possibly including shot fakes and things of that nature

- In principle, it should be able to give very fine path controls to the motor controller (i.e. it can make the mallet move in an arc if it wants)
- But also I (Thomas) plan to make the motor controller pretty robust so also the RL agent would **ideally not overly micromanage the motor controller**

Questions:

1. How can we make sure that our RL agent respects our kinematics limitations (e.g. max velocity, playing field constraints, etc)?
2. What kind of data is actually useful to the motor controller? In theory, the RL agent can produce a new position/velocity command every dt, but is this actually what we want?

Motor Controller

Our motor controller's goal is to physically execute the motions requested by the RL agent.

Given a current mallet position/velocity and a desired position/velocity at some future time, the motor controller will use a **parametric spline method** to compute a smooth motion that doesn't break our hardware.

Questions:

1. How can we make the motor controller and the RL agent work together, so that the RL agent doesn't make any **"unreasonable demands"** of our hardware?
    a. It would be unreasonable for the motor controller to bend to the RL agent's every whim at every dt. For instance, sudden direction changes are simply not possible, no matter how badly the RL agent wants it to happen.

Clearly, **the interaction between the RL agent and motor controller is the most critical part of our software package**.

RL Agent – Motor Controller Interaction

Possible solution: the MC only receives “critical” data points from the RL agent, i.e. moments where contact between mallet/puck are expected to happen (either on offense or defense) or at certain interpolation points along a path that the RL agent wants.

- This way, the RL agent only has to “know” kinematics, while the motor controller figures out the rest.

Related idea: the RL agent feeds the motor controller a “time series” of future position/velocities that can be dynamically adjusted, so the motor controller is largely “enslaved” by the RL agent at a high level but still calculates its own splines and velocities.

It may be possible for the RL agent to be trained while connected to a mock motor controller, which takes in the RL agent’s commands and returns whether or not the intended motion is actually possible.

- This would prevent us from losing in a real match, where the RL agent might want to do something impossible, the motor controller says no, and then we have no plan.
- If this feasibility signal is built into the training, then the RL agent can develop a backup plan ahead of time (using the time series approach) and we are never screwed.