Air-hockey Robot

SW503

30th of January 2020

Introduction

Results from the Project

- We implement the controllers to match their respective models as much as possible.
  - We expect the performance scores to match the models.
- Our experiments show that the block-rate is a lot lower.
  - The model shows a block-rate of 67.2%
  - Experiments had a block-rate of 24%
- It must be a difference between the model and the implementation.
  - Non-representative model.
  - Incorrect implementation.

Non-representative Model: Detection is Error-prone

- The object detection relies on colour, which we expect works well with the contrast of the puck.
  - However, sometimes the puck was never detected.
- Making the model representative of the environment:
  - Include detection failures in the model.
  - Use a different detection implementation.

Non-representative Model: The Camera Grid is Warped

- The Pixy series of cameras have slight fish-eye lens.
  - This means that the positions received do not have linear correspondence between the assumed positions of camera and mallet.
- Making the model representative of the environment:
  - Correcting the warp with a mapping from warped grid to straight grid.
  - Including the warp in the model.
  - Using a camera with less warp.

Non-representative Model: Camera and Table Coordinates Desyncs

- The camera coordinate system is lined up once, and never calibrated again.
  - Shifting could make the robot worse because every prediction is offset.
- Making the model representative of the environment:
  - Calibration can be done more often.
  - Model the calibration issues.

Incorrect Implementation: Measurements Lost in Translation

- Model uses both centimeters, and Pixy detection cells.
- Implementation uses just detection cells.
  - The table size was then incorrectly translated to detection cells from centimeters in the implementation.
- Making the implementation more faithful to the model:
  - Correctly translating centimeters to detection cells in the implementation.

Incorrect Implementation: Missing Steps

- Trying to move the mallet at top speed immediately, results in the stepper motor missing steps.
  - The electromagnets activate faster than they can pull the axle.
- Making the implementation more faithful to the model:
  - Implement the acceleration curve as modelled.
- Making the model representative of the environment:
  - Use a stronger power supply and removing the modelled acceleration curve.
  - Correcting the model to include missing steps.
  - Moving the mallet at a slower rate.
    - Now the implementation of the mallet becomes unrepresentative of the model.

Revising the Robot

The Revision

The Goal: We want to block air-hockey shots via an autonomous robot.

Figure 1: An Air-hockey table

The Single-axis Robot Design

- Single axis is enough to block the puck if enough information is gathered, and the mallet moves fast enough.
  - We can gather information with the help of a camera.
  - The mallet can move along a rail with the assistance of a stepper motor.

Figure 2: The single-axis robot design.

The Camera

- Pixy cameras come with built-in object detection via color recognition.
  - This allows for the transfer of recognised objects instead of pixel data via a serial connection.
  - The detection resolution is 320 by 210 cells.
  - Each cell is around $\frac{1}{3}$ of a centimeter.
- Alternatively, writing our own object detection.
  - Greater control over algorithm used.
  - The usage of background subtraction.
  - Reduce misidentification of hands and mallets as the puck.
- The Pixy 1 captures 50 frames per second and the Pixy 2 captures 60.
  - The Pixy 1 can capture 6.25 pictures for the fastest shots.
  - The Pixy 2 can capture 7.5 pictures for the fastest shots.

The Motor

- The motor is controlled by specifying a direction and pulsing the motor every time it has to take a step.
  - To move faster, pulsing will need to be faster.
  - Counting steps gives the precise location of the mallet.
  - Requires pulsing constantly.
  - The mallet can be moved at $1.3\frac{m}{s}$.
- Alternatively, the robot could have used a DC motor.
  - It requires a different way of finding the mallet position.
  - It can be faster and simply requires specifying the direction to move.

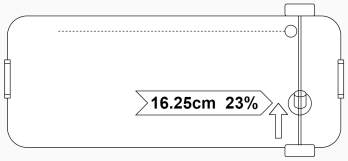

Figure 3: The amount of distance that can be moved during a shot

Designing a Controller

How Can We Play Air-hockey?

- We can program an algorithm that attempts to emulate human behaviour.
  - It is based on known succesful strategies.
  - It can only be as succesfull as the strategies the programmers know.
- Alternatively, we can use machine learning to learn an algorithm that can play.
  - It might find new ways of playing.
  - Internal preliminary experiments showed no positive results.

How Can We Emulate Humans Playing Air-hockey?

- We approximate the human process with three steps:
  - **Gathering information**: Detecting the puck.
  - **Processing information**: Predicting where the puck will end up.
  - **Reacting**: Moving the mallet to block the shot.

The Three Phases

Gathering information

- How can we find the information?
  - We find the puck via the Pixy camera by polling it via the serial connection.
- If we processed the image data manually.
  - The prediction info could assist in the object detection.
- Gathering data would work the same for both types of control.

Processing information

- Prediction is based on experiments.
- What we learned:
  - The puck does not move just as fast after rebounding.
  - The puck has an imperfect reflection.
- We assume:
  - The puck moves linearly.
  - Zero air resistance.
  - Zero surface resistance.
  - The position of the mallet is always known.
  - The camera can see the entire field.
  - The camera coordinates are directly translatable to positions on the table.
  - The gathered positions are correct.
  - The table is a rectangle.

Reacting

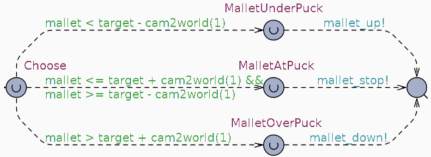

Figure 4: Three-step decision making

- We use simple three-step control to move the mallet.
- This assumes that if we brake, we will stay at the prediction.
- Other control methods might offer more accuracy:
  - PID control is almost universal.
  - Reverse the acceleration curve to find stopping point.

Finding the Controller

The Controller Design

- From the three phases we can create a controller that performs three steps.
- We create many variations of the controllers.
  - By further development
  - By combining controllers
- How can we know which variant to choose?
  - We can compare them and select the best performing ones.

Comparing Controllers

- We can compare by a quantitative value:
  $Utility = Blocking\ Rate - \frac{0.005}{1} \cdot Average\ Travelled\ Distance$
- The comparison value focuses on the two factors:
  - The blocking rate as this is the most important for achieving the goal.
  - The travelled distance to slightly prioritise more efficient controllers.
- There is no focus on:
  - Computation time.
  - Frequency of change in direction of the mallet.
- Collection of data about the controllers could be performed by:
  - Implementation and experiments to get the blocking rate and distance moved.
  - Modelling and simulations can figure out more data about controllers faster.

Modelling the Robot

Modelling the System in UPPAAL

- We can use hybrid timed automata to model the system.
- Divide the system into four subsystems:
  - Mallet
  - Puck
  - Camera
  - Controller

Mallet Model

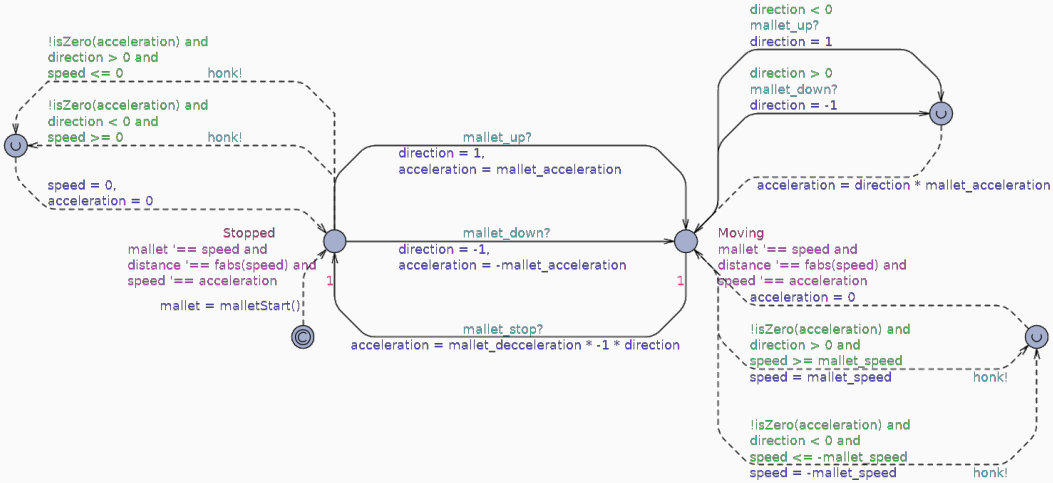

Figure 5: The model of the mallet.

Puck Model

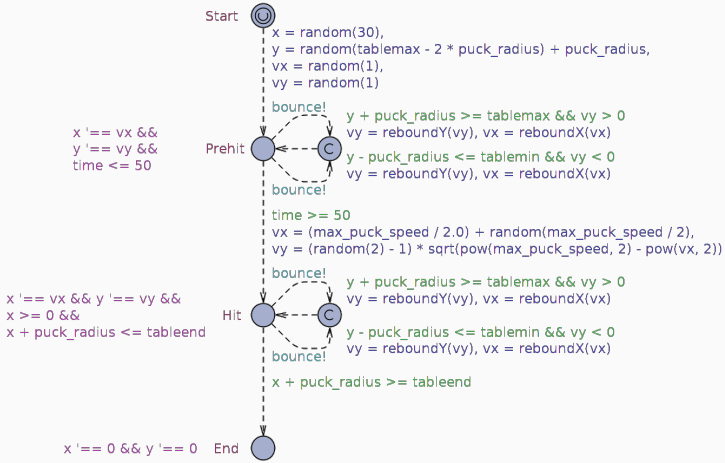

Figure 6: The model of the puck.

Camera Model

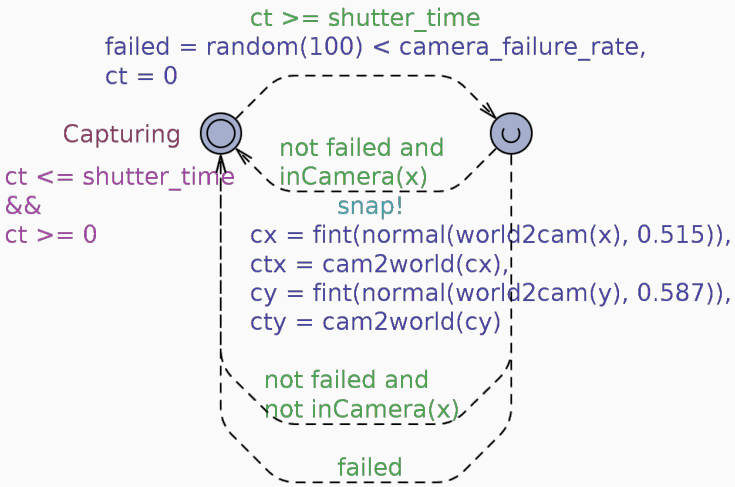

Figure 7: The model of the camera.

Controller Models

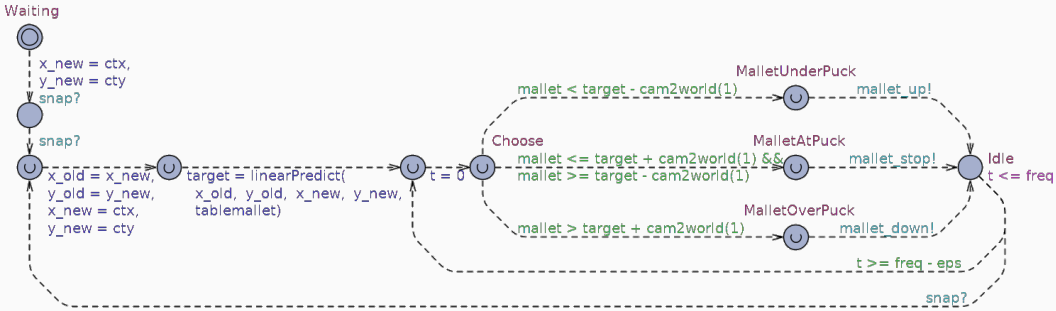

Figure 8: The model of the controllers.

Simulations

- Were simulations the right way to gather data?
  - We can run thousands of simulations quickly.
  - The model experiments can be evenly distributed.
  - Real life play will depend on the person playing with goals targeting the goal.
- The performance scores are easily calculated from the simulation-data.
- We can select the top performers for implementation.

Implementing the Controller

Choosing a Platform

- Control over scheduling
  - To make a scheduling guarantee more available.
- Processor speed
  - Processors that are too fast waste cycles and are more expensive.
- I/O ports
  - Control of the stepper require certain I/O capabilities.
  - Limit-buttons also require I/O.
- Price
  - Unnecesarily fast platforms waste money.
- We ended up choosing Arduino as the platform for implementation.

Implementing Controllers for Arduino

- The goal is to implement the model faithfully.
- We try to use the model code directly where possible
  - Some calculations in UPPAAL are done in C which allows direct transfer.
- Emulating the model transitions as close as possible.
  - Find pseudocode that matches transition choices.
- Controllers that are combined in the model, are also a combination in the code.
- The implementation has to worry about timings that are not in the model.

Timing Constraints

- We have to find what timings affect the implementation.
- There are three main environmental constraints
  - Maximum speed of the puck is $8\frac{m}{s}$ and the playing field length is $1m$ so a fast shot takes $\frac{1}{8}s$.
  - Stepper pulse rate of $325\mu s$ to $\infty\mu s$.
  - Camera serial transfer takes around $80\mu s$.
- The controller imposes real-time requirements as well.
  - Calculation times of the more advanced predictions are high.

Scheduling the Tasks

Task	Cost	Period	Deadline
R	50	>325	50
S	1600	none	20000

Task	Cost	Period	Deadline	Part
R	50	>325	50	50
S	1600	none	20000	<275

Table 1: Task data

- These tasks are not schedulable by a simple cyclic executive scheduler.
  - The cost of Sensing is bigger than the period of Reaction.
  - The Reaction has a varying period.
- We will need to make some changes to schedule it cyclically:
  - The Sensing task could be split into smaller parts and executed one part at a time.
  - The Reaction could run often and only when applicable, but it will require rounding the acceleration curve.

Designing the Scheduler

- We avoid preemption because the serial communication can not be preempted without redoing the communication
  - The data could be lost from the Pixy from a interrupted transfer.
- We use a custom scheduling algorithm.
- It can be defined as a variable-priority scheduler by:
  - It requires that we always choose the task with highest priority regardless of release status.
  - $P(Reaction) = 0$
  - $P(Sensing) = \text{Time until Reaction release} - \text{Cost of Sensing}$
- We can also see it as a fixed-priority scheduler.
  - Lower priority tasks will not be scheduled if higher priority tasks will be released during execution.
  - $P(Reaction) > P(Sensing)$

Verifying the Scheduler

Verifying the Scheduler Design

- How can we verify the scheduler design?
  - We can create a model of the scheduler and verify it.
- We make a timed automaton representing the scheduler.
- We can then use UPPAAL to verify the model using queries about the scheduler.

Model of the Task

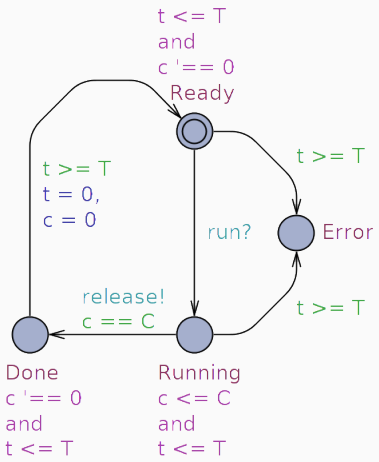

Figure 9: The timed automaton of the task

Improved Model of the Task

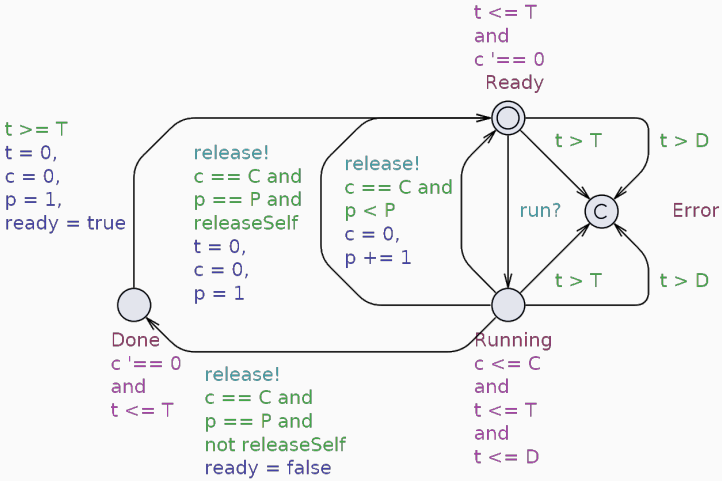

Figure 10: The timed automaton of the task

Model of the Scheduler

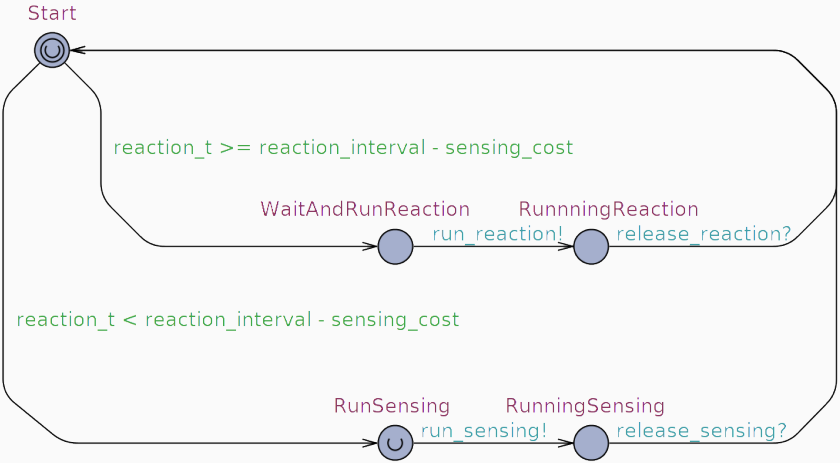

Figure 11: The timed automaton of the scheduler

Improved Model of the Scheduler

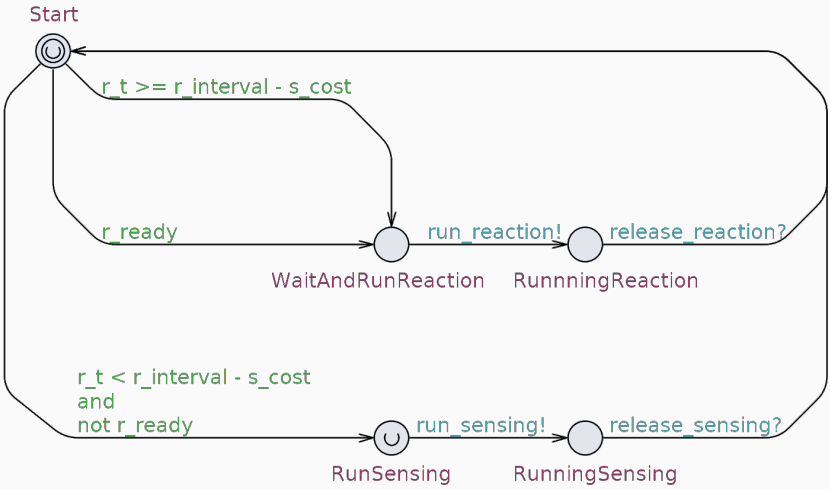

Figure 12: The timed automaton of the scheduler

Implementing the Scheduler

- UPPAAL shows that:
  - The deadlines are not broken.
  - The scheduler will not deadlock.
- To implement the UPPAAL model we work from pseudocode that emulates the model:

Algorithm 1: The scheduling algorithm

```
while running do
    if R is ready or R Release − C(Sensing) < 0 then
        wait until R is released;
        execute R;
    else
        execute S part;
```

- This is then turned into real code for the Arduino.

The Resulting Robot

Improvement results

- Now that the model and implementation has been changed to take our considerations into account.
- We redo the simulations, which shows a $48.2\% \pm 0.5$ catch-rate for the improved model
- Experiments with the implementation was conducted by doing two series of 100 shots.
- The experiments showed a 46% and a 42% block-rate for the improved implementation.
- The new results are a lot closer, which could suggest that:
  - The model fits the the real world better.
  - The implementation is more true to the model.

Demo

Appendix

Movement Requirements

- Moving the mallet to minimise distance with the mallet (Bangbang control)
  - Requires that the mallet moves just as fast as the transverse movement of the puck ($8\frac{m}{s}$)
  - Only requires a single known position for the puck.
- Looking at where the puck is and where it is moving to (Linear extrapolation)
  - Requires at least two known positions of the puck.
  - $\frac{1}{8}$th of a second across $70cm$ requires a speed of $5.6\frac{m}{s}$

The Top Speed of the Mallet

- The motor requires 400 pulses per rotation of the axle.
- One rotation is 170mm movement of the mallet.
  - 85 teeth with a pitch of 2mm.
- We have $\frac{1}{8}$ of a second for a fast shot.
  - 1m of playing field and $8\frac{m}{s}$.
- We can pulse every 325µs because of the controller and powersupply.
- This is ~384 steps for a fast shot.
  - $\frac{\frac{1}{8}\frac{m}{s}}{325\mu s}$
- So we can move 163mm during a fast shot.
  - $\frac{384}{400} * 170mm$