ECE8 Robotics Notes 10-20-22

- See file: PlanarPositionDiscreteTimeWithStepInput.m
    - Contains all the relevant information and examples for today's lecture
- Class begins with MATLAB
- Prof. makes several changes to this file over the course of the class

- Prof.'s 1st Question:
    - In code, how would we assign the desired location?
    - Answer: as simple as letting some variables be those values

```
px_desired = #

py_desired = #

Professors notation:

pxd = #

pyd = #
```

- MATLAB results:
    - Professor adds code like the one above to set the desired location "ending locations" for the robot
    - After plotting it we see that the path did not pass through the target
        - this means we missed our target
        - the remaining parts of this lecture are on how to fix our code to force the robot to find the desired location
        - the final code is uploaded to canvas
- Prof's 2nd Question:
- What can we do the code to make the robot get to the correct position?
    - student response: change the slope
    - Answer by Prof:
        - We CAN adjust the slope (aka the velocities)
            - this is adjusting the speed and direction of the robot (code) as we approach
        - However that would require that we know the angle at which we approach the desired location
- Prof's 3rd Question:
    - What if the robot is close to the target?
        - Answer:
            - we can adjust the allowed movements to stay within the target distance
            - this means adding (if-statements/conditions) to the code when it is near the correct values
            - we can also set a tolerance value

Handwritten Notes Lecture 9

- Today we will learn to design the input to 2 discrete-time model to reach a target location
    - this statement the professor wrote didn't make sense
- Recall from the previous lecture:
    - In MATLAB, we obtained the following position plot:
    - see prof. notes where he draws a graph of px vs py
        - where px is on the horizontal
        - py is on the vertical
        - red line begins at $(1,0)$ and moves upward to the point $(2,1)$
            - $(p_{x0}, p_{y0}) = (1,0)$
            - $(p_{xd}, p_{yd}) = (2,1)$
    - prof added two graphs to the right of the position graph
        - v_x vs k
        - v_y vs k
        - where k is our discrete time variable that increments in integer steps
        - where v_x and v_y are velocity in the x and y directions respectively
        - these graphs look like straight lines because they represent constant motion (constant velocity)
- Prof.'s 4th Question:
    - what does $k = 0$ represent and where is this time on these graphs?
        - Answer:
            - $k = 0$ is the initial time value, we always start at zero
                - This value is found in the positions: p_x vs p_y (starting position)
                - $(p_{x,k}, p_{y,k}) = (p_{x,0}, p_{y,0}) = (1,0)$ "formal notation"
                    - positions at initial time $k = 0$
- professor proceeds to connects how the 2 velocity graphs correspond to the position graph
    - he draws lines connecting them
        - a flat line in v_x with it being flat at zero means the robot is only moving vertically
        - a flat line in v_y with it being flat at zero means the robot is only moving horizontally
- To use the Planar Model
    - Where planar = plane model, i.e., working only in 2D e.g. $(p_{x,k}, p_{y,k})$ (2 coordinates)
    - Model:
        - $p_{x,k+1} = p_{x,k} + \Delta * v_{x,k}$
        - $p_{y,k+1} = p_{y,k} + \Delta * v_{y,k}$
    - With given desired positions (p_{xd}, p_{yd})
    - We want to find inputs $v_{x,k}$ and $v_{y,k}$ so that $(p_{x,k}, p_{y,k})$ reaches p_{xd}, p_{yd} at some discrete time k
        - Another way to say this is: for some time/value k we get
            - $p_{x,k} = p_{xd}$
                - "Current x position" = "desired x position"
            - $p_{y,k} = p_{yd}$
                - "Current y position" = "desired y position"
        - aka: The Robot reached its goal position
            - E.g.: the drone reaches the height you want it to reach
- Goal for today:
    - figure out a way to reach a particular point with MATLAB

Solve the Q: reaching a particular point

- prof. returns to MATLAB
    - Difficult to write what he is doing but in summary:
        - in the code we have several if-statements that control the speed in x and y directions
            - we kept adjusting their values and conditions to change the path of the robot
            - This was done while checking the 2 velocity graphs that generate
        - The professor kept adjusting until our path hit the target
        - New issue:
            - the robot hit the target but continued to moving – we want it to stop moving
- Option: Simulation Horizon
    - In our code this is represented by
        - # of steps "N"
    - We can reduce the number of steps to make the robot stop operating
    - The professor reduced the number of steps to make the robot stop operating once it reached its target

Issue with adjusting the Event Horizon

- This method does not work for any initial condition
- It only works for the specific one we fiddle around with
- Meaning this code is limited to one trial

- Prof.'s 5th Question:
    - Consider wanting to walk to the door across the room, What would we need to do in order to get to a particular location, like the door?
    - "what is the 1st thing to do to get to the door?"
    - Answer:
        - examine/assess the desired location
        - plan the route
        - this is called: Determining the **Error signal**

Error signal

- Error signal of the door:
  - Example:
    - determine the error between where **you are** relative to **the door**
      - **"** I am 5 ft from the door" – sample 1
      - **"** I am now 10 ft from the door" – sample 2
        - (Alert!, you're error is larger, you're farther from the door)
      - " I am now 2 ft from the door" – sample 3
        - (Error is decreasing, you're closer to desired door)
      - "I am at the door" – sample 4
        - (Error is zero- You are at desired door)
  - Zero Error:
    - if error is zero then this means we are at the desired location.
  - Error:
    - this will tell us/robot how well we're doing at reaching the location
    - the **larger** the error the father way we are and the worse we are doing
    - the closer the better and this is a **smaller** error
  - Equation for error in our code:
    - error in x = desired x location – current location
      - $err_x = p_{xd} - p_x$
      - $err_y = p_{yd} - p_y$
  - The professor added the necessary code that accounts for this sort of error
    - the result is now going to tell us how much error we encounter as we loop through the computation
    - We need to:
      - turn this into a form of feedback
        - we will take the error and tell the loop to adjust before each new step
        - so the output of each step is sent to the next starting input
        - "a full loop / feedback loop"
- Student Q:
  - Why did he pick N =600 vs N = 60
  - Prof. Answer:
    - N represents how many samples we do, or steps we take fully
    - when N is too low we may not get to the position but the direction will show it tried
    - with too many N it would still get to the location but we will have more sample points
      - more dots on the graph
    - Conclusion: the number of N can be anything, just need it to be enough to reach the desired point

Summary in Prof. Handwritten notes

- The class is RECORDED?? I HAD NO IDEA
- During our initial work in MATLAB we tried to find $v_{x,k}$ and $v_{y,k}$ to steer us towards the desired points
    - $p_{x,k}$ to p_{xd}
    - $p_{y,k}$ to p_{yd}
    - though, this will require multiple tries. More importantly this will require that we recalculate $v_{x,k}$ and $v_{y,k}$ if either p_{xd} , p_{yd} or p_{x0} p_{y0} change
    - This was using only the Event Horizon

Introduction to feedback

- See Block/boxed diagrams in Prof.'s notes
    - "Closed loop system"
        - because the code we used "shown with lines/wires in the drawing" feeds back the current positions $p_{x,k}$, $p_{y,k}$ to the inputs
    - consider the planar model – this is placed in the "controller box"
        - $p_{x,k+1} = p_{x,k} + \Delta * v_{x,k}$
        - $p_{y,k+1} = p_{y,k} + \Delta * v_{y,k}$
    - and, desired positions
        - p_{xd}, p_{yd}
- we compute the input velocities $v_{x,k}$ and $v_{y,k}$ using the error signals
    - $ex = p_{xd} - p_x$
    - $ey = p_{yd} - p_y$
        - Professor used ex and ey in his code
        - in the above notes they were err_x and err_y

Changing the code to have one with for-loop and one with while-loop

- The professor included files that show how to do the same operation with both a for-loop and while-loop
- Tolerance:
    - A parameter to set a level of closeness
    - Say we want a very small error basically zero
    - we can set a variable called $tol = 0.0001$ "desired closeness"
        - e.g.:
          while ex >= tol && ey >= tol % while the tolerance is too large keep running
            - where && means and (both must apply)
                - review Boolean logic for more insight
                    - A && B means:
                        - work but only if both A and B are true
    - ex and ey can be negative depending on position so we wil need to use
    - $abs(ex)$ and $abs(ey)$ to remove negatives from ex and ey to just compare the size to tol