Robot Intelligence **Name: smile group: Bradley Je**
Fall 2023
Midterm Exam
8/31/2023
Take Home Exam - Due Before Class on Canvas 11/3/2023

This exam contains 24 pages (including this cover page) and 8 questions.
The total number of points is 160. Graduate students must answer some additional questions. Undergraduates answering graduate questions will be given additional points and a feeling of satisfaction from attempting difficult things (probably).

Any programming/code artifacts associated with this exam should be submitted along with the final PDF to canvas. Please put your entire submission in a single .zip file.

This exam will cover topics in motion, planning, sensor processing, and ethics.

Grade Table (Quick view to see the breakdown)

Question	Points	Score
1	20	
2	20	
3	20	
4	20	
5	20	
6	20	
7	20	
8	20	
Total:	160	

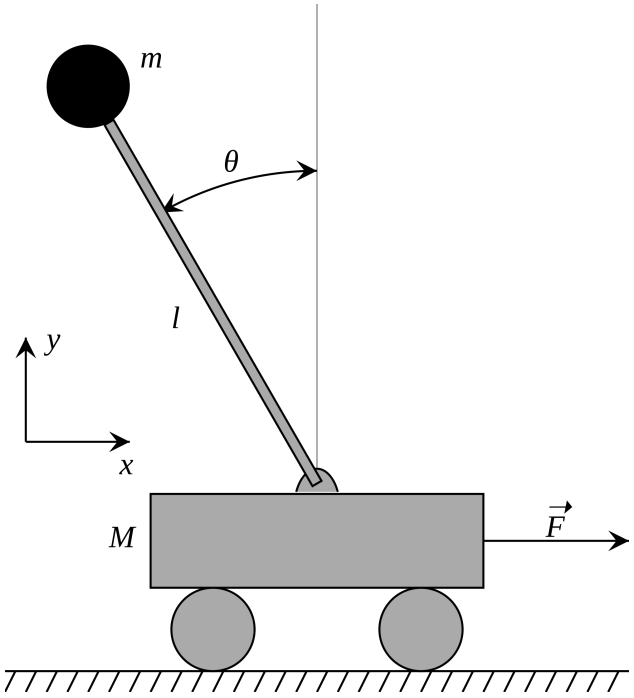

Figure 1: Assume that the mass $m = 0.5kg$, $l = 0.8$, $M = 3.0kg$

1. (20 points) Balancing a Pole

   A cart and pole suddenly appear before you, and you feel compelled to answer burning questions you have always held about these systems.

   (a) (5 points) Describe the equations of motion that govern this system

   There are two major equations that govern the friction-less model of the system:

$$\ddot{\theta} = \frac{gsin\theta + cos\theta(\frac{-F - m_p l\dot{\theta}^2 sin\theta}{m_c + m_p})}{l(\frac{4}{3} - \frac{m_p cos^2\theta}{m_c + m_p})}$$

$$\ddot{x} = \frac{F + m_p l(\dot{\theta}^2 sin\theta - \ddot{\theta} cos\theta)}{m_c + m_p}$$

   Where:

   - $\ddot{\theta}$ is the angular acceleration of the pole
   - g is the gravitational constant
   - θ is the angle of the pole from the normal of the floor
   - F is the force being applied to the cart in Newtons
   - m_p is the mass of the pole
   - l is the length of the pole
   - $\dot{\theta}$ is the angular velocity of the pole
   - m_c is the mass of the cart

- $\ddot{x}$ is the acceleration of the cart along the x-axis

The first equation solves for angular acceleration of the pole which is necessary for determining whether the pole is rising or falling. The second equation solves for the horizontal acceleration of the cart which determines whether the cart is speeding up or slowing down.

(b) (10 points) Generate code for a controller that would be able to keep the pole balanced in the air.

Code 1: Cart Pole Code

```
def agent_theta(state):
    x = state[0]
    x_dot = state[1]
    theta = state[2]
    theta_dot = state[3]
    if theta_dot > 1:
        return 1
    if theta_dot < -1:
        return 0
    return 1 if theta > 0 else 0
```

(c) (5 points) What is the maximum angle that my pole can fall to before it cannot recover if max Force $F = 7N$?

To solve this problem we made a few assumptions. Our first assumption is that the starting angular velocity is 0. This assumption will allow us to calculate the greatest possible recovery angle, given ideal conditions.

After making this assumption we graphed the first equation from part a and found the X intercept between our area of operation $(0 \leq \theta \leq \pi/2)$

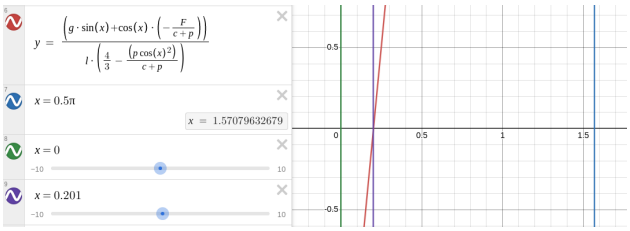

Figure 2: Desmos Graph where $y = \ddot{\theta}$ and $x = \theta$

Based on this, any $|\theta| < 0.201$ will be recoverable.

2. (20 points) Open-Loop Control

   I would like you to use an Ackermann model of a robot that is $35ft$ long and $10ft$ wide (this is a standard US Firetruck) and run a few simple experiments with it. Please upload your resulting figures and Python (or other) code:

   (a) (5 points) Make a list of commands (at 2Hz) that will allow this robot to traverse along the edge of an $18m$ radius circle. The robot starts off in the center of the circle (0,0). Plot the resulting path (x, y) and trajectory (x, y, and angular velocities). Assume a constant velocity of $8m/s$. **Graduate Students: You cannot leave the circle's border.**

   Here are some high level instructions that we would give our robot:

   1. Drive for 3 seconds with $\alpha = 0.9273552132019242$
   2. Drive for 12.5 seconds with $\alpha = 0.5880602936363463$

   Here are the figures:

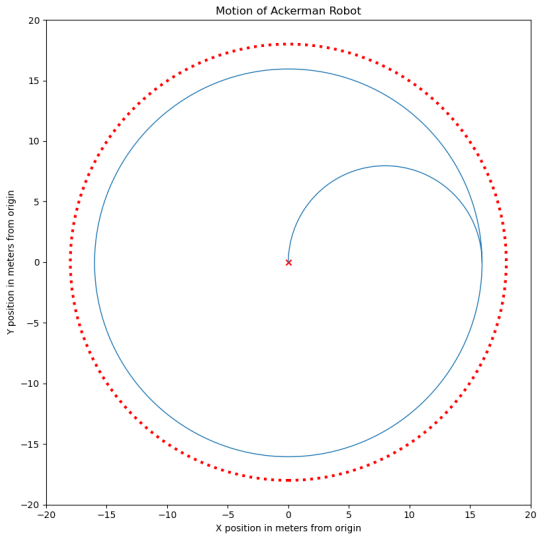

Figure 3: Path of Ackermann robot

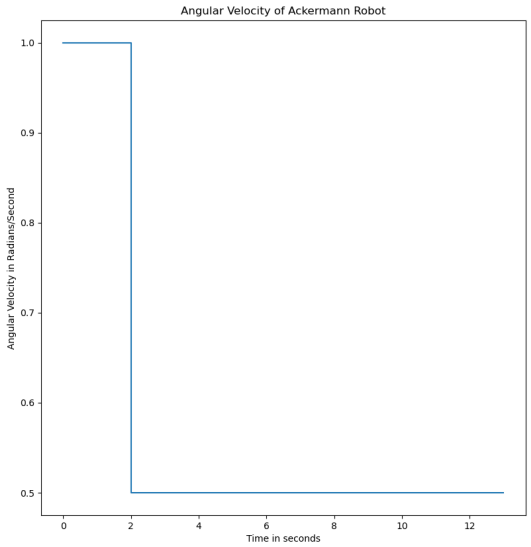

Figure 4: Angular Velocity of Ackermann robot

(b) (5 points) Do the same as the above for an equivalent skid-steer vehicle. **Graduate Students: You still cannot leave the circle's border. Mwahaha**
See Figure 5 and 6 for graphs.

(c) (10 points) Your Skid Steer vehicle (with the same dimensions as described for the skid steer), is driving on a circle of radius 9m. Assume that you begin on the edge of the circle. Calculate the positional error with our computational approximation using the forward Euler method, as referred to in the course notes and illustrated in Reading 2, eq. 1.6. Graph the errors and computing time for three different time-steps ($\Delta t = 1, 0.1, 0.01$), error is defined as the absolute distance between the expected (from equations) to real x, y position (defined analytically) over time.

Brief notes on what this problem is about: Imagine that you are moving a semi-truck and you have GPS positions (somewhat accurate) and an internal prediction model. We want the internal models to be updated based on "ground truth" information to build better estimates of the vehicle motion over time.

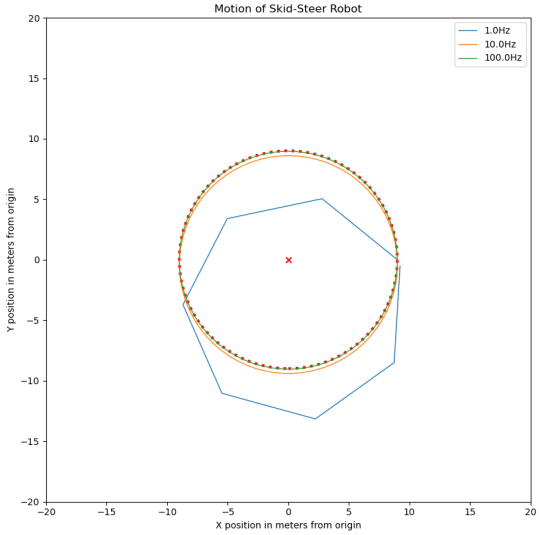

Figure 5: Path of Skid-Steer robot at various refresh rates

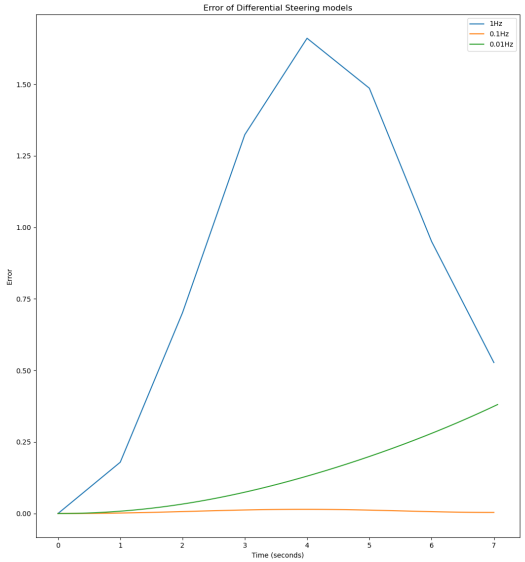

Figure 6: Error of Skid-Steer robot at various refresh rates

(d) (10 points) **Graduate Student Question** Compute part c again where road frictions are much lower (due to rain or ice). Slip causes your tires to respond very differently. Assume that your theta is $\theta_{actual} = \theta(1 - 0.08)$ and velocity is $v_{actual} = v(1 - 0.04)$.

Comment on the differences you would expect if the Vehicle was loaded with water (increasing the mass by 2.5x).

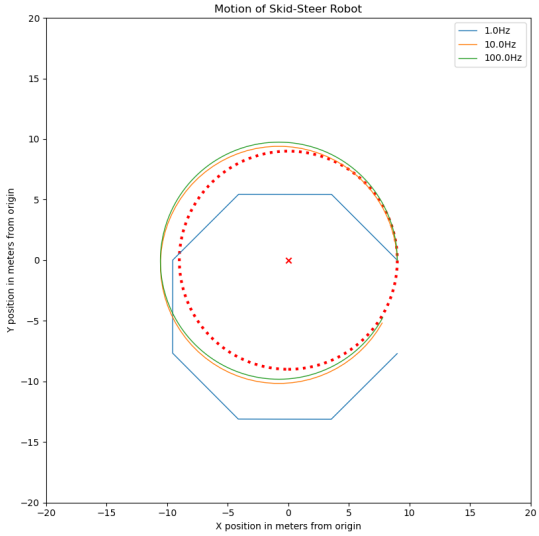

Figure 7: Path of Skid-Steer robot at various refresh rates

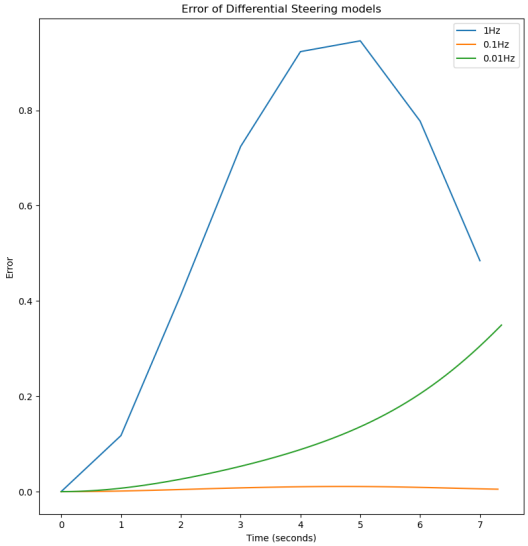

Figure 8: Error of Skid-Steer robot at various refresh rates

If we were to increase the mass by 2.5x we would expect the velocity and angular velocity to further decrease. On a slippery surface a robot with a greater inertia

would be more difficult to control and would take an even wider turn.

3. (20 points) (a) (10 points) Implement the monocular velocity detection algorithm (`https://github.com/ZhenboSong/mono_velocity`. Measure out the distance to an object and calibrate to ensure a stable baseline.

   Comment on the distance to three additional objects. How accurate are the distance estimates? Take a video recording of a moving object and run the algorithm on it, compile a graph of the estimated velocities. Comment on the results, particularly on the errors or perturbations in the readings taken. (May want to plot the FPS and minor perturbations at each time step.) Detail the merits of the algorithm and explain in a few paragraphs how the monocular velocity detection algorithm works.

   This algorithm was very challenging to implement and with the little to no documentation offered in the repository, we were not able to get it to work. Much time was spent trying to build and run the project. After many attempts at running the code and deciphering the error messages to get the correct dependencies installed, we were able to get the code to start running without dependency issues. The next major problem that we encountered and were not able to overcome was how to setup the code to run with our own video inputs to complete this problem. Unfortunately, the README in the repository did not offer much help on how to run it and we were not able to figure it out on our own. Fortunately, we can still describe the algorithm a little bit and how it works.

   Monocular velocity is a very simple and cost-effective way to determine distance and velocity of objects. It is cost-effective because it only requires a single camera to gather the input data. This makes it a lot more available and versatile since cameras are all around us these days and it can be used in all types of applications from robotics to surveillance.

   The monocular velocity algorithm works by utilizing an algorithm called optical flow which analyzes the pixels and their changes in various ways, such as intensity, from frame to frame. This is used by an optical flow algorithm to compute the apparent motion that the pixel may have had. This algorithm also uses feature tracking to help determine distance and velocity of an object. Feature tracking identifies key points in a frame or image (such as the corner of an object) and matches it to the same feature in the next frame or image if found. The change that occurs in the key points from frame to frame is used to calculate the motion of the object. These two algorithms combined are what make up the majority of the monocular velocity algorithm to detect the distance and motion of objects using a monocular camera.

   (b) (10 points) Implement the Monocular Odometry algorithm (`https://github.com/alishobeiri/Monocular-Video-Odometery`) on a pi or laptop. Record a video of a walk in a large loop (indoor or outdoor), and do this circuit several times. Comment on the positional error after several loops. Submit the video and position graph from the algorithm and your comments on what happened and why you believe this occurred. Detail the merits of the algorithm and explain in a few paragraphs how the

monmocular odometry algorithm works.

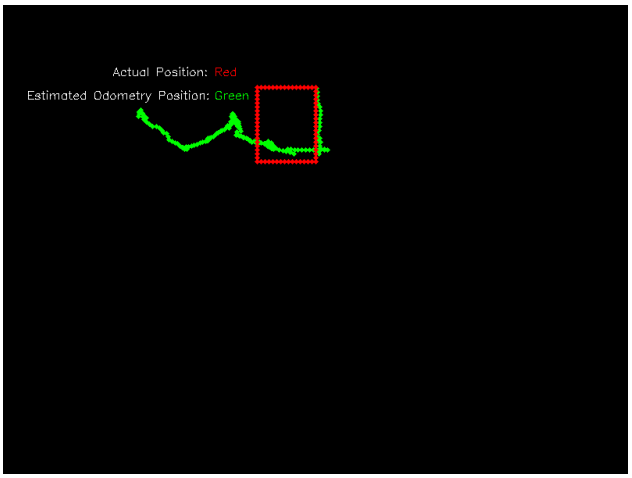

Figure 9: Path from visual odometery algorithm (green) vs expected path (red).

The positional error after several loops was quite large. The algorithm performed a lot better while moving straight than while turning. This was to be expected of the algorithm since it is generating a path based purely on the key points in each frame and the difference in these key points between each frame. During the turns of the video, there are a lot less key points in frame since the camera is close to a wall with little to no features to give the algorithm key points to base its path off of. This also likely had a significant impact on the performance of the algorithm.

This algorithm identifies key points in each image (representing one frame of the video). These key points are anything easily identifiable by running some simple computer vision algorithms and filters on the image. Easy to find features include corners or sharp turns in outlines of an object. These key points are then found in the following image. The algorithm uses the difference in location of these key points to calculate the change in position from frame to frame. These calculations are added up over all the images and a final path is produced by the algorithm (shown in green in the image).

4. (20 points) Localization and Mapping Simultaneously (LAMS)

   (Please use the proper term SLAM)

   (a) (5 points) What are the inherent problems of relying solely on a single sensor for localization? Comment on each of the following:

   - IMU - This will only give you internal change in position without knowing where you started at or the position of objects around you. Drift will also accumulate over time resulting in a growing error relative to distance traveled.
   - Sonar/Ultrasonic - This will only give you the distance to an object but not what the object is or if the object is moving. It can also be disrupted easily with noise and interference.
   - GPS - This will only give you your current global coordinates without telling you where other objects are nearby. This can also be obstructed while indoors or in other areas that may block signals such as a canyon which will lead to inaccurate and slow results.
   - Monocular camera - This will only provide a 2D image with no depth so you can see what objects are in front of you but not how far and also without knowledge of scale. This can also be heavily affected by changes in light and other weather conditions.

   (b) (10 points) Implement the ORB SLAM 3 `https://github.com/UZ-SLAMLab/ORB_SLAM3` on a laptop or pi with your choice of monocular or stereo vision as your hardware may permit. Map an area of a minimum of 150 sq ft indoors. Save the final map and submit it as part of this question. Comment on the accuracy of the solution.

   Unfortunately, due to similar issues experienced while trying to complete problem 3a with the monocular velocity repository, we were not able to get the ORB SLAM 3 algorithm working to complete this problem as well as the graduate question in this section. There were a lot of issues encountered when trying to setup the environment correctly and get all the dependencies installed with the appropriate versions that worked with each other correctly. Multiple versions of Ubuntu were installed and many packages and modules were installed over the past few weeks in the hopes of finally getting the recipe right to run ORB SLAM 3. Eventually, we believe that we were able get all the dependencies installed correctly, however, similar to the monocular velocity question, we were not able to get it to run correctly with our own input video or live camera feed.

   Based on the example shown in class, ORB SLAM 3 is not a perfect mapping tool. Similar to the other video based detection algorithms (monocular velocity and monocular odometery), there are a lot of common issues that can lead to inaccuracies in the results. Blurry frames, quick turns, lighting conditions, and insufficient feature points can all have a large effect on the accuracy of this algorithm.

(c) (5 points) Autonomous vehicles often rely on several cameras, Lidar, and Radar. See the diagram below and comment on the required range of sensors. Look up the cost of each sensor in this package (give a final figure with links). Assume one very capable lidar or several directional lidars.

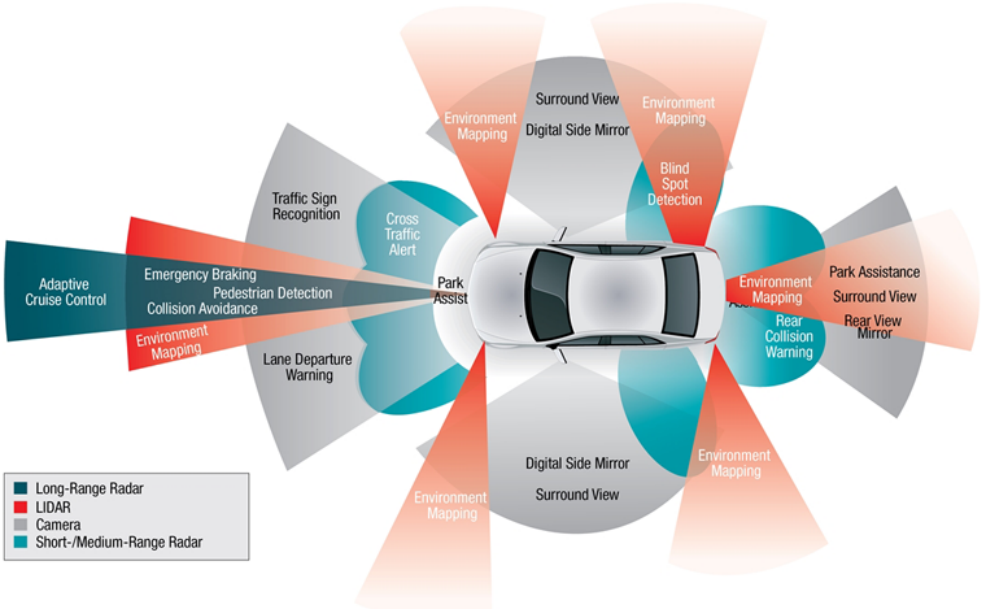

Table 1: Sensor Package Cost

Sensor	Unit Price	Units	Total Cost
Long-Range Radar	$23,950	1	$23,950
LIDAR	$1000	6	$6000
Camera	$10,000	4	$40,000
Short-/Medium-Range Radar	$19,950	1 (package)	$19,950
Total			$89,900

https://www.ti.com/sensors/mmwave-radar/automotive/products.html#sort=1130;desc&(AWR2944)
https://cdn.neuvition.com/media/blog/lidar-price.html
https://www.eautotools.com/Autel-ADAS-s/2478.htm?searching=Y&sort=2&cat=2478&show=120&page=1
https://www.ti.com/sensors/mmwave-radar/automotive/products.html(AWR1843)

(d) (10 points) **Graduate Student Question** Use the same OrbSlam algorithm on an outdoor scene of 300 sq ft. Attempt to wander in a circuit several times (at least 3) and record the path taken as a ROS bag. Submit the bag file along with comments on the performance relative to the indoor environment in part b. Explain why the algorithmic performance differs.

As mentioned in part b, we were unable to get ORB SLAM 3 to work properly. We

can still, however, comment on the potential performance differences when running this algorithm outdoors compared to indoors. The lighting conditions from time of day, weather, and shadows in an outdoor scene can heavily affect the performance of ORB SLAM 3. Indoors you can have consistent lighting at better levels compared to outdoors where you can have very harsh lighting from the sun. This makes it harder for the algorithm to detect changes. Outdoor scenes could also have a lot less textures and features available for feature mapping. Another potential impact on performance outdoors is determining the scale in an image with potentially large open spaces. Moving objects and occlusion can also be an issue if there are people or animals moving about in the scene. These are all examples of potential impacts on an outdoor scene compared to an indoor scene where these conditions would be less likely to make an impact in performance.

5. (20 points) Control and Reinforcement Learning

   (a) (5 points) Explain three merits and three demerits of using reinforcement learning for mechatronic systems.

Merits

1) **Adaptability**: Reinforcement learning can adapt to changes in the system dynamics or external disturbances by continuously learning from its environment. This adaptability makes it more robust to uncertainties and changes that might occur during system operation.
2) **Data-Driven**:Reinforcement learning can optimize control strategies based on real-world data. Instead of relying on human expertise to manually design controllers, reinforcement learning algorithms automatically optimize actions to maximize rewards. This can be especially beneficial for complex systems where deriving a manual control strategy is challenging or impractical.
3) **Complex Task Learning**: Reinforcement learning is particularly effective for tasks that are hard to define with explicit rule-based approaches. For instance, in robotics, teaching a robot to balance or navigate uneven terrains can be achieved more effectively through reinforcement learning, as it enables the robot to explore different strategies and learn from its mistakes.

Demerits

1) **Safety Concerns**: During the learning phase, reinforcement learning algorithms can take actions that are not necessarily safe for the system. This can result in damage to hardware components or even pose risks to human operators. While there are techniques to ensure safe exploration, they add another layer of complexity to the RL process.
2) **Sample Inefficiency**: Reinforcement learning typically requires a large number of samples (interactions with the environment) to learn a policy. For mechatronic systems, this can translate to a lot of wear and tear, especially if the learning process involves many suboptimal or even potentially harmful actions.
3) **Complexity and Training Time**: Implementing and training reinforcement learning algorithms can be complex, especially for high-dimensional systems. It often requires specialized knowledge in machine learning. Additionally, training might need extensive computational resources and can take a significant amount of time, especially for deep reinforcement learning approaches.

   (b) (5 points) Draw a diagram for reinforcement learning and controls and contrast the two

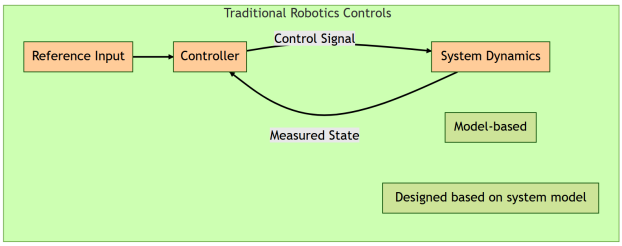

Figure 10: Diagram for Controls

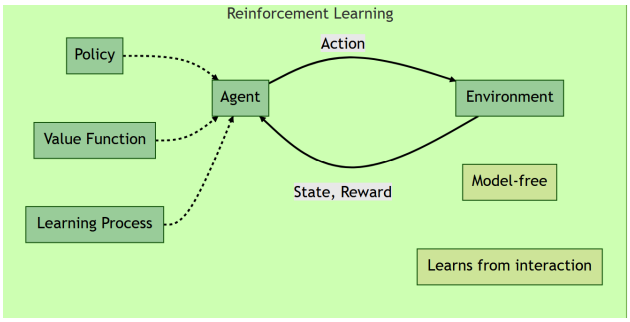

Figure 11: Diagram for Reinforcement Learning

(c) (10 points) Use OpenAI Gym to create a trained reinforcement learning model for the cart and pole problem (CartPole-v1).

`https://gymnasium.farama.org/environments/classic_control/cart_pole/`

Alter the structure so the motion to the right is 50% more aggressive than moving to the left (ie: velocity to the right is faster by 50%). Record the video of this and submit the edited code and recording.

Now implement the Frozen Lake scenario (`https://gymnasium.farama.org/environments/toy_text/frozen_lake/`) and adjust the environment to work on a randomly generated 8x8 map. Compare the training performance for this map against a similar 8x8 map that has a random chance (2%) of the agent randomly dying of cold.

See Problem 5, Part C, README.md to see graphs and analysis comparing these.

(d) (10 points) **Graduate Question:** Implement this example of a 2D running robot (cheetah) and adapt the code to penalize hip motor movements faster than $\pi \frac{rad}{s}$. `https://gymnasium.farama.org/environments/mujoco/half_cheetah/` A Virtual Machine that was made with VMWare is available on Canvas for you to use for this question if you have trouble installing Mujoco.

6. (20 points) Human Emotions

   (a) (10 points) **Using the FER library in python, example in the following link:** `https://towardsdatascience.com/the-ultimate-guide-to-emotion-recognition-f` build a system that classifies human emotions and validate on video 1 and 2 from this repository: `https://github.com/rjrahul24/ai-with-python-series/tree/main/07.%20Emotion%20Recognition%20using%20Live%20Video` Generate plots of predicted emotions over time for both videos.

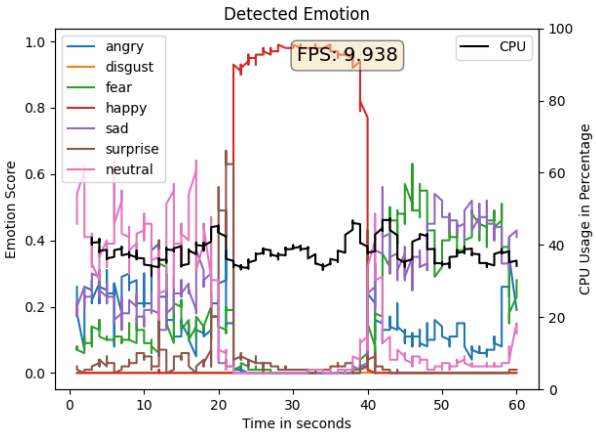

Figure 12: FER Data for Video 1

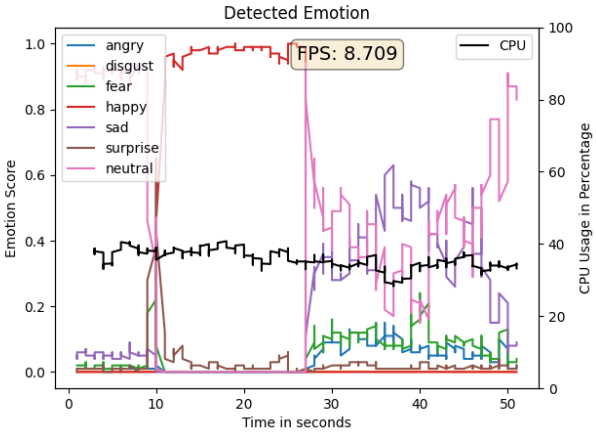

Figure 13: FER Data for Video 2

   (b) (5 points) **Do the same as the above with a video feed from your webcam. Set your software up to allow video feed or a pre-recorded video. (In essence, make faces at yourself and make sure that your service works). Submit a short faces-recording along with the script that can read live webcam streams. Implement this on the Raspberry Pi (can use the**

stored video from a different device) and detail the speed (FPS) and CPU usage between your pi and your standard use computer you are using (hopefully not a pi). Note: please add the specs of the comparison device.

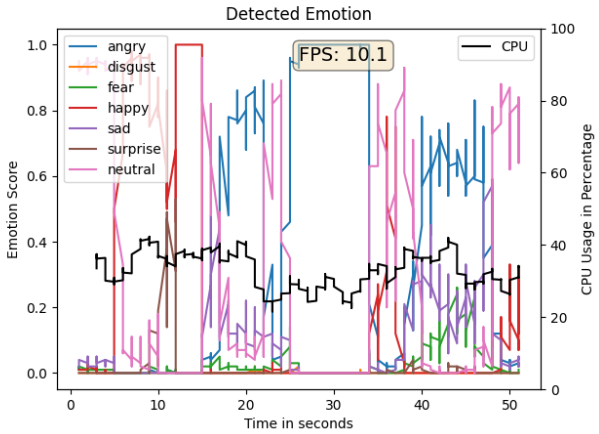

Figure 14: FER Data for Video 2

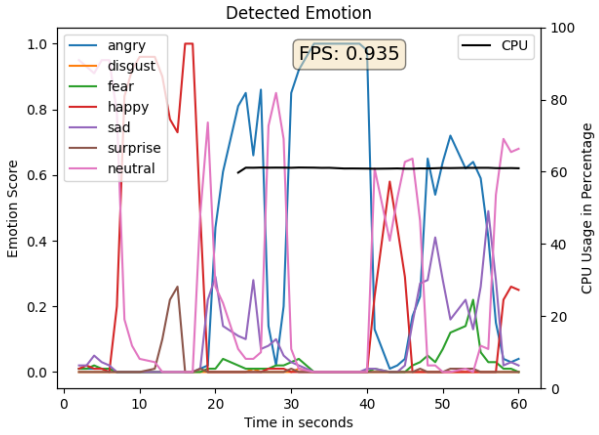

Figure 15: FER Data for Video 2

A note on the CPU Usage: different operating systems

Table 2: FER Video System Spec Comparison

Spec	Dell Precision 5560	Raspberry Pi 3B
CPU	11th Gen Intel(R) Core(TM) i7	Quad Core 1.2GHz Broadcom BCM2837
RAM (GB)	32	1
GPU	NVIDIA T1200 Laptop	N/A

(c) (5 points) **Using the source material in the article (`https://www.section.io/engineering-education/nlp-based-detection-model-using-neattext-and-scikit-lea` build a rudimentary NLP based emotion analysis tool to analyze the data found in: `https://www.kaggle.com/datasets/praveengovi/emotions-dataset-for-nlp`**
Please see 'problem6/textAnalyzer.py' in our code repository.

(d) (5 points) Detail (with a diagram) a software stack for a robotic system that can take advantage of these two tools. Write out the software architecture (UML, Flow chart, ect.) and talk about the purpose and usefulness of the software stack.

There are many potential uses for an emotionally responsive robot. A patrolling police robot can alert authorities when it hears people in distress. A self-checkout point-of-sale machine with such a capability can bring useful insight regarding trends in customer satisfaction. Both of these examples sound dystopian, but they also would be extremely useful in identifying issues that may have otherwise been ignored. Below shows a high-level representation of the physical and software components required in an emotional responsive robot intelligence stack.

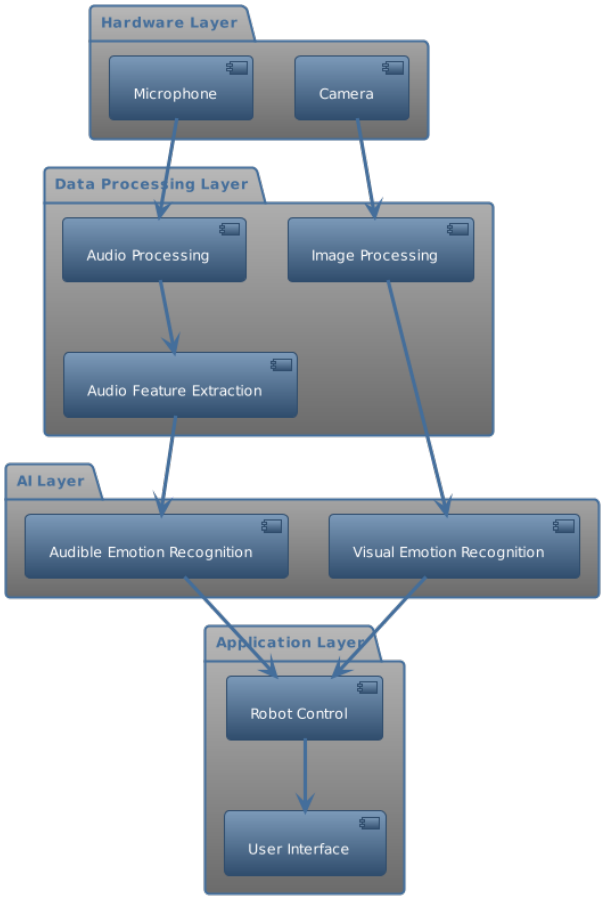

Figure 16: High-level Software Architecture of Emotionally Responsive Robot

(e) (10 points) **Graduate Question: Build a robot reaction stack that can per-**

ceive human emotion and identify parts of the scene to respond quicker or differently than a standard motion planner. Detail the software as in part (d) above.

We will be using an example of a robot designed to assist in a hospital setting. As such this robot will need cameras, microphones, audio and image processing each with feature extraction, the emotion recognition as well as other sensors such as an infrared or heat sensor, to help indicate body temperature and collect initial data for the nurses and doctors. Figure 17 displays how the robot would interact with the user. Figure 16 shows almost an exactly what we need for this robot. The only thing would be to add an infrared camera at the hardware layer, and processing for that at the data processing layer. The AI layer would need processing for the data processed from the infrared camera. This would be sent to the robot control and user interface with the resulting information.

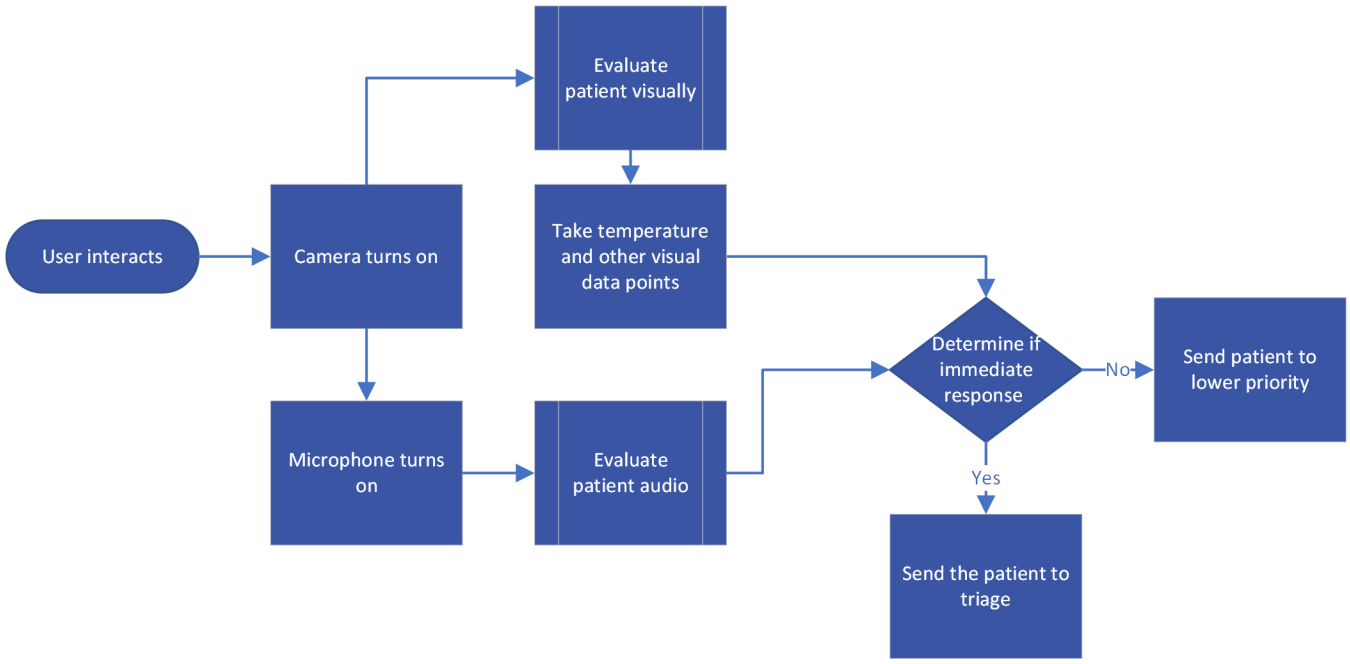

Figure 17: Hospital robot to user interaction

7. (20 points) Motion Planning

   (a) (10 points) **Implement an A* based planner (an example is found here:** `https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2`**) and compare it's results with the Djikstra, A-Star, RRT-Star, Bi-Directional A-Star, and the Breadth First Search Planners from this github.** `https://github.com/AtsushiSakai/PythonRobotics/tree/master/PathPlanning`**. Create a table comparing the average cost of the path found over 10 iterations along with the time to convergence.**

Table 3: Path-finding Algorithm Comparison

Metric	Our A-Star	Djikstra	Their A-star	RRT-Star	Bi-Directional A-Star	B
Time (seconds)	14.85	3.04	3.49	66.11	3.47	2.
Cost (euclidean distance)	50	53	45	59	46	45

After you have implemented your planner and compared it answer the below questions.

- **Which planner provided a path with the lowest cost on average?**
  - It is a tie between their A-star and the Breath-first Search.
- **Which one found a path the fastest on average?**
  - Dijkstra.
- **After comparing your planner to these five other ones is there anything you would change in your planner to help it converge faster or find a path with a better cost?**
  - Currently, our planner only expands to neighboring nodes expands to 4 neighboring nodes at each iteration. It will be much more efficient if it were to look at 8 directions, including diagonally.
- **Which planner appears to be the best overall? Which planner would you use for a robot in a complex environment?**
  - According to our collected data, the comparison A-star and the Breath-first search have the same average cost, yet the Breath-first was quicker by an average of approximately 5 seconds.
  - For complex environments, the rapidly exploring random tree (RRT*) would be a good choice. It is the only planner listed that works in continuous space, meaning can be much more flexible with mapping.

   (b) (10 points) **You are tasked with developing a path-planning algorithm for a newly designed planetary exploration rover. The robot has advanced localization capabilities and uses an internal map to navigate. This internal map is provided from satellite scans and includes terrain elevation data, represented on a scale from 1 to 4. The robot experiences increased difficulty when transitioning from lower to higher elevations compared to moving on flat terrain or descending.**

   Given this elevation grid as an example:

4	3	2	2	2	2	1	1	1	1
3	2	2	LAKE (can not be crossed)	2	1	1	1	2	1
3	2	2		2	1	GOAL	3	3	1
2	2	1		4	2	4	4	3	1
2	1	1		4	4	4	4	2	1
1	1	1		3	4	3	2	2	1
1	2	2	2	2	3	1	1	1	1
START	1	1	1	1	1	1	1	2	2

Traversal costs 1 for traversal between cells. However, an additional cost (+1) is incurred when moving uphill. Downhill motion reduces costs (by -0.5) for downhill. Assume motion is valid in any of the eight adjacent directions.

Select an appropriate path-planning algorithm for this scenario. Discuss how you would modify the algorithm to optimize for the robot's path, taking into account the effects of terrain elevation. Note that the "optimal" path in this context refers to cost, not necessarily the shortest distance. Show what this optimal path should look like on the map given. Show what an optimal path would look like if the normal traversal between cells is 2. Define reasonable assumptions about how different elevations impact the robot's speed, keeping in mind that ascending should be slower than other types of movement. Provide a general description of your chosen algorithm and demonstrate how it would navigate the given elevation grid. Your answer should clearly articulate your approach to optimizing the robot's path, given the unique constraints of its capabilities and the terrain.

We designed a modified A* Star path-finding algorithm because A* is guaranteed to find the optimal path, and it is complete; a solution will always be found if one exits. For our implementation, cost g has been modified such that it increases more when there is an increase in altitude between nodes and vice-versa. The heuristic cost remains important in keeping the algorithm efficient and goal-driven.

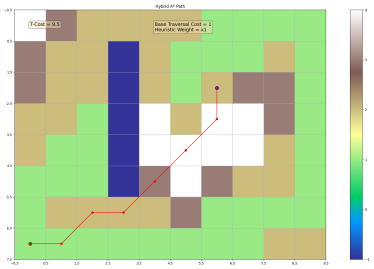

Figure 18: Path Found with Our Hybrid A* with a Base Traversal Cost of 1

Show what an optimal path would look like if the normal traversal between cells is 2.

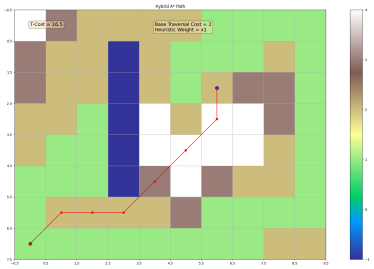

Figure 19: Path Found with Our Hybrid A* with a Base Traversal Cost of 2

Changes in the elevation of the terrain can greatly impact the robots ability to move forward. When climbing in altitude, more power is required to counteract the additional gravitational forces required to move; ascending is a drain on energy consumption as well as speed. As seen on the two figures above, the change in the base cost was not enough to significantly change the path.

(c) (10 points) **Graduate Question: Implement a skid-steer (4 wheel) vehicle model for a robot 33 inches wide, by 40 inches long for the Hybrid A* algorithm. Change the scenario so both your A* and Hybrid A* have the same obstacle field. Compare and contrast the final trajectories and graph these. Write the algorithmic differences between the two planners and how the kinematics are used to constrain the Hybrid A*.**

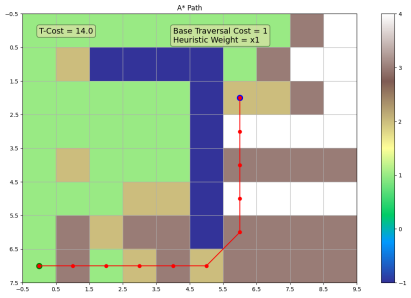

Figure 20: Path Found with Our Regular A*

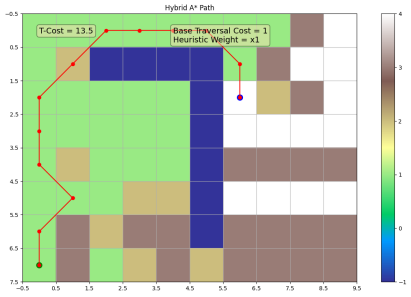

Figure 21: Path Found with Our Hybrid A*

The dimensions for a skid-steer vehicle are irrelevant given that each grid is big enough for it to rotate freely. The only difference between our A* and Hybrid A* algorithms is that instead of simply increasing in cost for every node traveled, the cost function also take into account the added cost of going uphill and the benefit of going downhill. This takes the kinematics of the system into account. Looking a the two figures above, our regular A* algorithm found the fasted path in terms of length. It fails to take into account the changes in terrain and instead goes through a more difficult path.

8. (20 points) Object Detection (Please put the classified images as part of your submission)
   (a) (5 points) Classify the 100 images in the Store category from `https://web.mit.edu/torralba/www/indoor.html` using any image classification algorithm (not YOLO).

   We classified the images using the ResNet model. `https://pytorch.org/hub/pytorch_vision_resnet/`

   (b) (5 points) Do the same with comparing 2 YOLO versions:
   - YOLO v8 (`https://github.com/ultralytics/ultralytics`)
   - YOLO v5 (`https://jordan-johnston271.medium.com/tutorial-running-yolov5-machi`

   Note: You are free to use the Ultralytics version, but I wanted to expose you to the process for simplifying networks so you can deploy on hardware.

   Discuss the differences in terms of accuracy and speed.

   In general, the v8 algorithm performs better than the v5 algorithm in the sense that it can more accurately detect things in the image. Usually the v5 algorithm missed things that were mostly occluded whereas the v8 was able to detect them. However, the v8 algorithm was ever so slightly slower than the v5 algorithm.

   This test was run on a Laptop running Arch Linux, with an i7-8565U @4.6GHz and 8GB of RAM.

   See Table 3 for specific performance metrics.

   (c) (10 points) Deploy a YOLO model on a Raspberry Pi. What is the effective performance you get on the same datasets? Build a chart of performance showing the speed (FPS) and CPU usage between your pi and the standard use computer you are using (hopefully not a pi). Note: Please add the specs of the comparison device.

Table 4: YOLO performance

Model+Hardware	avg FPS over 5 runs	CPU peak
YOLOv8+Laptop	8.06	56%
YOLOv5+Laptop	8.30	64%
YOLOv5+Raspi4	0.44	300%

   A note about the pi, I was only able to get through 65% of the dataset before the pi began to overheat.

   (d) (10 points) **Graduate Question:** Implement Detectron (`https://github.com/facebookresearch/Detectron`) and test the image segmentation of the objects you are interested in. Give metrics on relative segmentation/classification quality comparing Mask R-CNN, faster R-CNN, and RetinaNet.

   These models tended to better find smaller (and more) objects than YOLO. Mask R-CNN has the added ability to be able to tell exactly how much of the image is a part of the identified object (instead of just a bounding box), however Faster R-CNN performed better at classifying objects. RetinaNet was able to classify objects with a much greater granularity, but with much lower accuracy.