A (Mars?) Rover

Jayden Chan Imtihan Ahmed Laura Petrich

Introduction

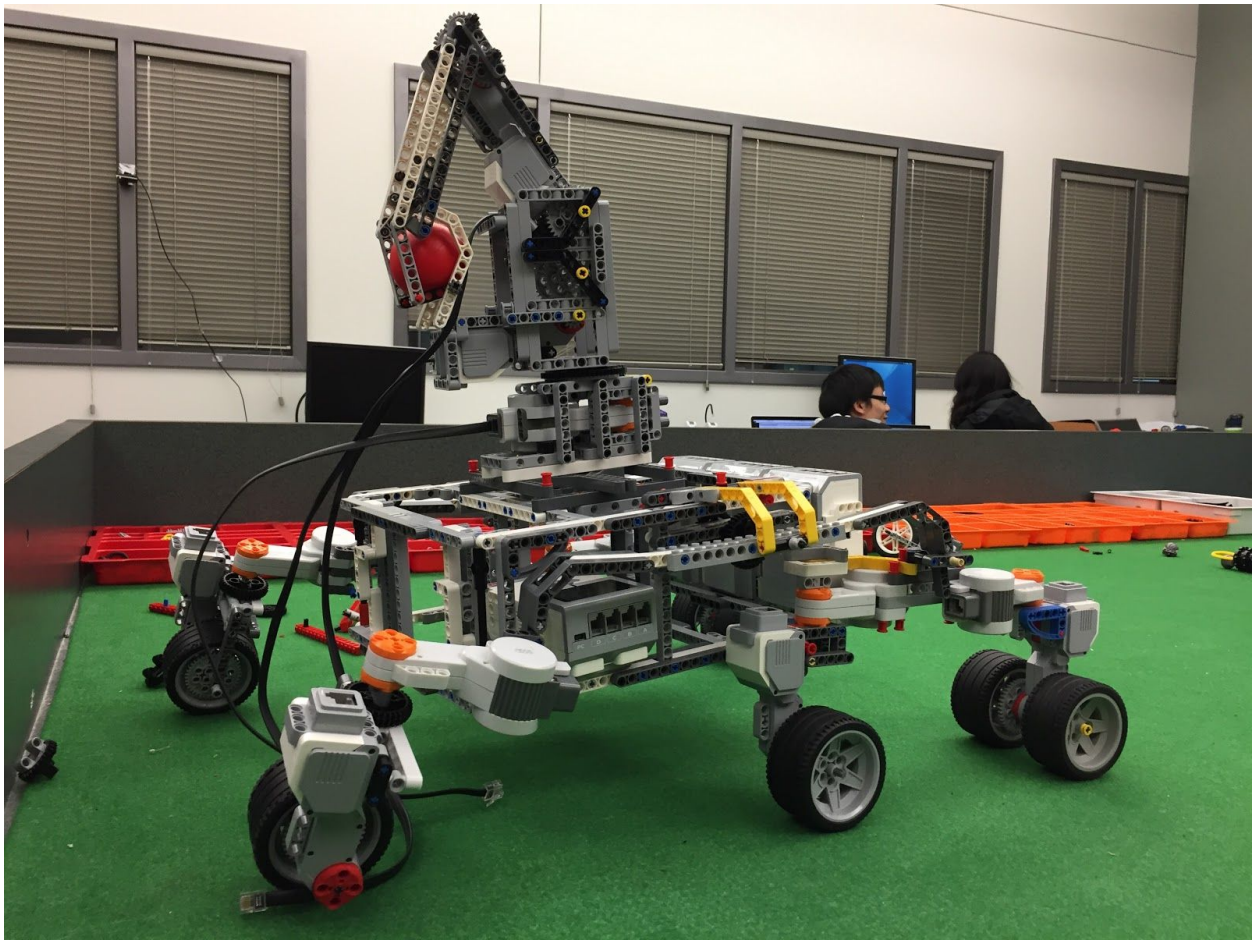
Robotic developments have been instrumental to space exploration by advancing scientific breakthroughs, and fostering mankind's ambition and curiosity to explore the universe. The Mars Rovers are the epitome of robotic development doing just this. We were inspired by the Mars Rovers to model and adapt certain features using Lego Mindstorm Kits and running Lejos on the EV3 platforms. Throughout the evolution of the project we encountered many obstacles and challenges, some which could be overcome, and some which caused us to adapt the scope/expectations of our project accordingly.

Motivation

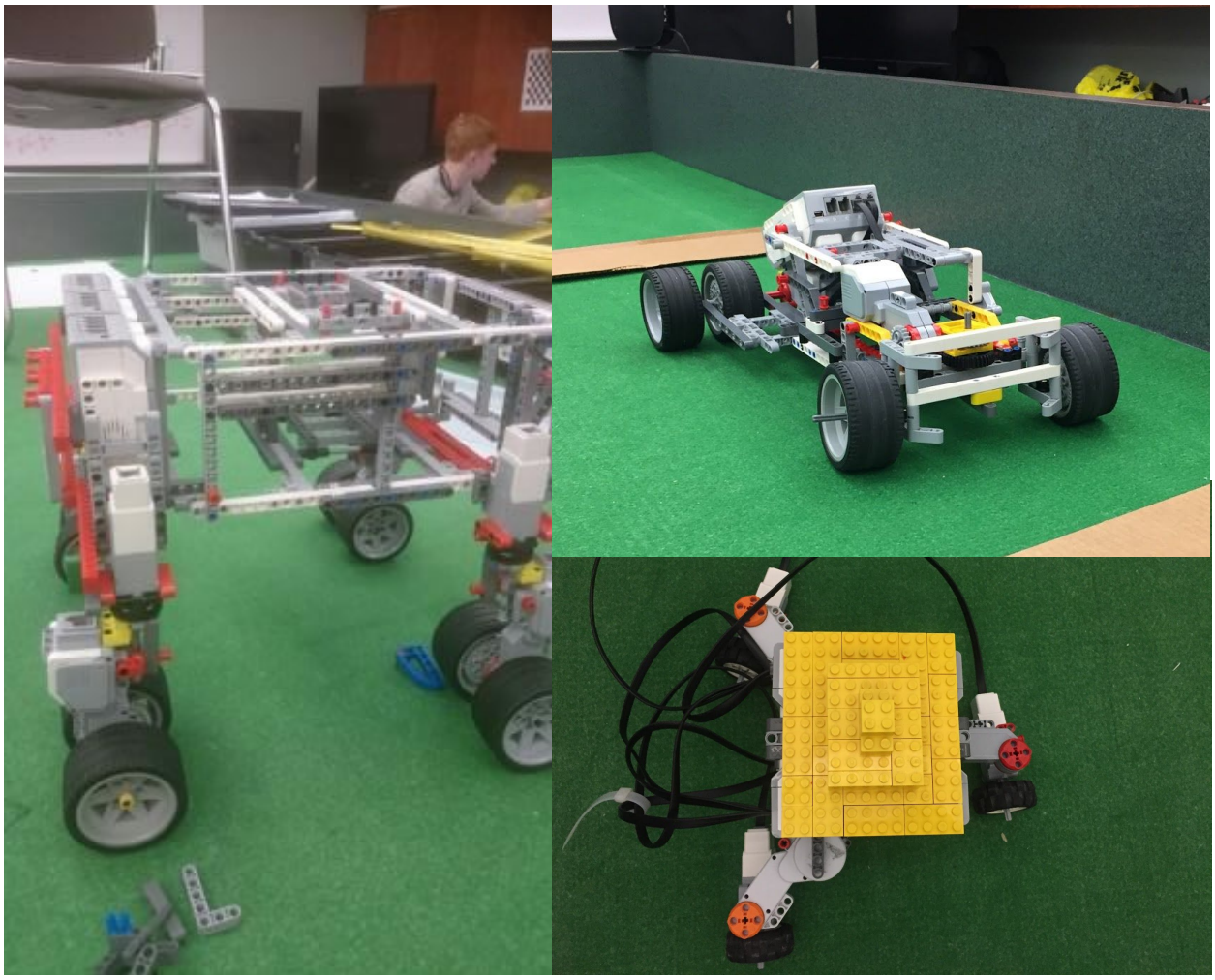
The Canadian International Rover Challenge and the University Rover Challenge motivated us to base our project on concepts from the Mars Rover.

Evolution of the Robot (Jayden)

Originally, we had lofty goals for this project, but we soon realized that some implementations would not be feasible given the equipment and time constraints. The project grew and adapted as our knowledge on the subject increased and obstacles arose. The rocker bogie suspension system is one of the distinguishing features of the Mars Rovers. It’s impressive ability to travel over obstacles and traverse challenging terrain make it the optimal (so far) choice for mobile navigation on Mars.



Our first prototype embodied this system but proved to be too unstable for the size we wished to achieve, and the lego available for building materials. Our second prototype used a 4 wheel configuration, and directly attached the wheels to the frame. Each wheel had it's own motor, with the front and back wheels each having an additional steering motor which allowed the vehicle to still turn in place . In exchange for more stability, the rocker bogie system was removed, and motion in the z axis for each wheel, independent of the others, was sacrificed. We soon realized that the new configuration made the vehicle too tall and would potentially topple over, when the arm attached, while making sharp turns. The arm was also unable to reach the floor to pick up objects. This led to prototype 3, which was a big change from the original system, and looked more like a car than a Mars Rover. The new prototype implementing Ackerman Steering, was stable, and had the advantage of being close enough to the ground for the arm to pick up objects. The prototype had a lot of potential, but as the visual servoing and path planning aspects of the project can into effect, it was realized that this prototype had a turning radius too large for the available workspace. We wanted to be able to demonstrate the implemented object avoidance and dynamic path planning, and in order to do so the robot needed to move swiftly in the workspace. This led to the prototype 4, the final robot, which implements a holonomic based system. Three small wheels are controlled by two motors each, resulting in a 360 degree range of motion, in-place rotation capabilities and the ability to swiftly navigate the entire workspace as needed.



Communication (Jayden)

Our communication protocol was designed to be easy to use as well as adaptive to the scope of the project. It is designed to be able to dynamically initialize motors as the computer requires it to. It is also able to set the speed of these motors or specify an angle to move the motors to. Using this interface we can do all the algorithmic thinking and calculation from our pc rather than using the EV3 hardware for computation. This speeds up our program as well as reducing the strain on the EV3. Communication is handled by two different files. multiPC.java and ev3.py. multiPC.java is to be run in the LEJOS environment on the EV3 and ev3.py is designed to be imported into any python application to communicate with the EV3s which are loaded with multiPC.java.

Some challenges involving the communication includes figuring out how to communicate using Java and Python sockets. How to initialize these sockets to work properly with a bluetooth PAN. How to set up a PAN with two or more EV3s. As well as reading and writing action commands properly.

Visual Servoing (Imtihan Ahmed)

Visual servoing was an essential component of our project, and we needed a reliable framework that would be able to properly ascertain object and spatial data so that our path planning and object avoidance behaviours would work well. Without an object being detected in the space, the robot may believe that no objects are in its path and attempt to go through it.

There are certain conditions we would have liked for our visual servoing to be effective, and are as follows:

1. A dual-camera input stream with live video feed
2. Originally, the processing was supposed to be done on an onboard raspberry pi mounted on the robot itself.
3. The cameras were to be mounted on the robot.
4. We wanted to be able to classify up to 8 objects in the space, with one or more of them being target objects we wanted the robot to move towards and the rest as obstacles.
5. Using computer vision techniques to relay real-distance metrics to our path planner.
6. If time permitted, a face classifier that would allow the robot to move towards one of the individuals.

The inspiration behind these objectives were done in spirit of the Mars Rover. As we expect communication lines between the rover and Earth to be time delayed and unreliable, we wanted our implementation to also be somewhat self-sustaining behaviourally and computationally independent so that live communication between an external device and the robot was unnecessary.

Points 1, 2, and 3 were quickly modified, as we observed that weight and budget constraints would impede on our ability to have a battery-powered raspberry pi and a second camera mounted properly on our robot. As our robot base went through iterations, it became clear that effectively stabilizing the robot with a lot of weight through the equipment would be extremely constraining, so we decided to do the computation for visual recognition on a laptop.

For visual recognition, we decided to use the popular openCV3 and numpy libraries available for python 3.6. This choice allowed the visual servoing information to easily communicate with our path planning module as that was also written in python, leaving the interfacing difficulty only from the path planning to the actual kinematics on the ev3, which was written in Java.

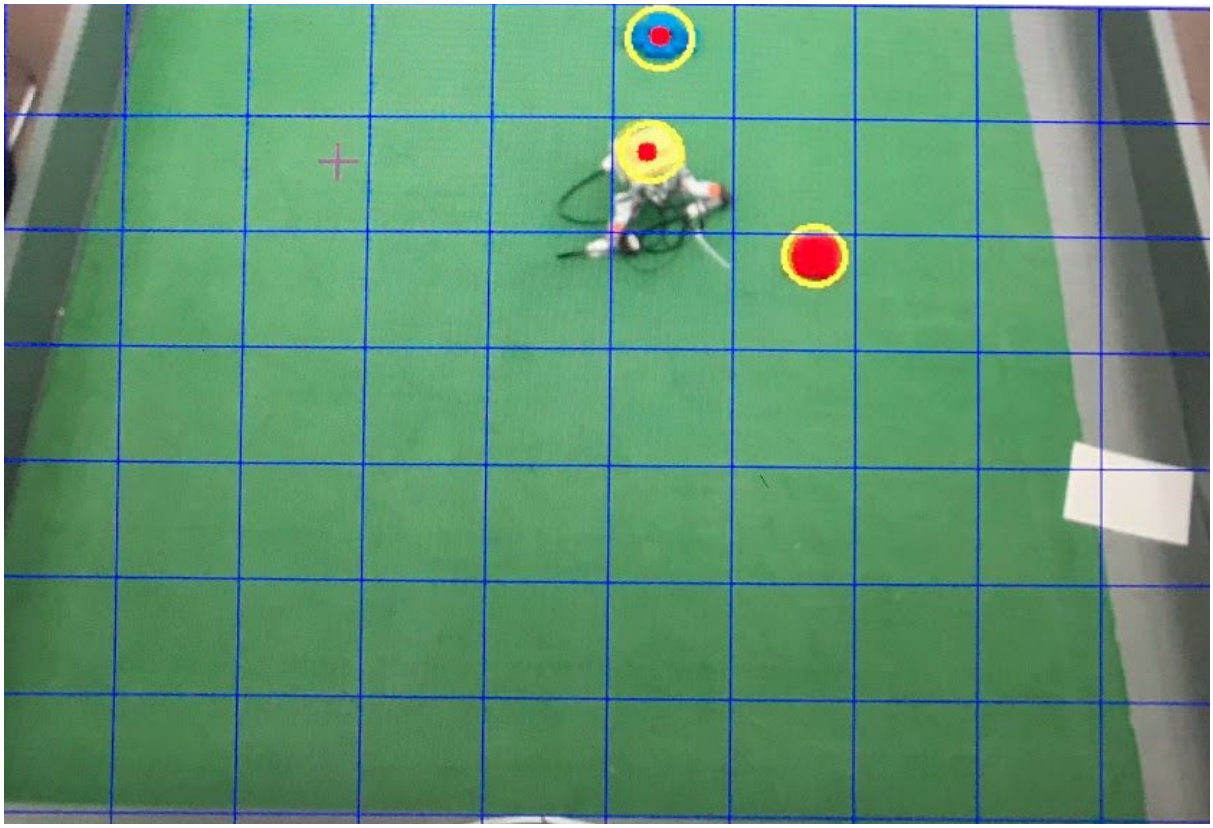
The second issue came in the form of object classification. After training a single object classifier on a tissue box (which is a relatively simple object albeit with some non-uniform artwork on the box) using 10 positive sample pictures and a few thousand negative sample images for training, the classifier performed extremely poorly. Slight orientation differences or lighting changes made the classifier detect no objects, and considering the length of time that was required to train a classifier for a single object (around 5 hours) and the added time required to create positive samples and get

negative samples, we decided that object classification with a classifier we trained would not be feasible for this project. However, as a final test, we decided to use a pre trained Harr face classifier to see if we could put faces on objects to detect, but that also ran into issues with orientation and detection.

Thus, object classification was finally done using contrast detection with openCV3. After testing with RGB value ranges, we discovered that it is simpler to use HSV colour space to define the ranges in which colours will be detected. We decided to have 4 colours to detect objects with: blue, yellow, red, and black. However, the workspace shadow around the edges and reflectivity of materials made it very difficult to distinguish objects that were black. Defining area ranges for the colour and object did not relieve the issue as artifacts of various sizes were detected in the black range. Therefore, the black colour was dropped and we proceeded with red, blue, and yellow objects. We decided to constrain the blue object to be within a specific range so that no artifacting would affect the detection. With a green floor surface the blue colour was detected accurately within almost any lighting condition. For red objects, which were to represent obstacles, we were more liberal with sizes, as we wanted to account for objects of various sizes as obstacles. Multiple red objects could be detected and fed into our path planning algorithm. The yellow object was constrained in size and worked consistently well enough to detect, thus we used this colour to determine the robot location in the workspace.

This final version of the visual servoing technique was effective at object recognition, and we could move our focus to determining position. The initial approach with the single camera and a reference object of known size to determine relative distances worked well in mathematical terms. We found the focal length of the camera used for this project, and one of the red balls in the laboratory, and attempted to determine distance and position metrics in this way. However, the program is good at detecting an object, but has oscillations when it comes to finding the contour of the object. Thus, we had significant errors that prevented positioning to be easily described in this method. The errors seem to stem from random environmental variables, and was not skewed or repeatable in any way, thus we could not adjust our measurements in this way. Perhaps using a higher quality camera with more resolution or dynamic range will alleviate the issue somewhat, but that was beyond our budget constraints. An alternative that may be effective for the next time is a depth sensor paired with a camera, where the depth sensor can determine the distance metrics to the objects recognized by the camera. However, that would again become a budget constraint.

Thus, to reduce error as best as possible, we decided to have the camera in as much of a top-down view of the workspace as we could achieve, and then translated coordinates in the pixel space to real space for each object found in the image. The program is designed in such a way that once the objects are recognized, a user input is requested (by pressing “q”) to feed the data to the path planning algorithm for further processing. This is done to ensure that at least some objects are detected, and mostly a diagnostic feature (sample picture below).



Finally, the position coordinates were received, and as we required grids to perform path planning, the visual servoing program intrinsically adapted these coordinates to grids, and then sent them to the path planning algorithm.

Path Planning (Laura)

We started off with the goal of implementing optimal path planning and real time obstacle avoidance. Ideally, the robot would initially calculate a predetermined path based on optimal cost estimates and update the shortest path as new obstacles are detected. All that was needed for initialization was the start coordinates of the robot, and the end goal coordinates (either gathered visually, or fed into the program if the goal location was hidden from view). After researching path planning algorithms that are/were implemented in the industry, I decided to implement a version of D*lite in our program. This is because it would allow our robot to dynamically update its path as new obstacles are detected without incurring high costs of recomputation. It avoids these high recomputation costs by storing the location of each obstacle and grid costs and only recalculates values for nodes that need updating as obstacles are encountered. It assumes that any unknown “free space” is traversable, and only sets the cost to infinity if an object is detected. Only then does it recalculate a new path from previous/newly calculated costs. In doing so, it avoids recalculations for all values. Computationally, D*lite has a higher initial cost, but as the workspace grows larger, it is computationally more efficient than other path planning programs, such as A* or Life Planning A*.

Although the program should theoretically be able to recalculate according to dynamically placed obstacles, we were unable to test it due to visual servoing and time constraints. As currently

implemented, it assumes that the robot takes up one node space. The goal, and each obstacle also cover one node each, even if this may not physically be the case. Under these assumptions, the program was able to successfully determine the correct path that accumulates the minimum traversal cost. It assumes that any diagonal movements incur a cost of 1.4, all other movements incurred a cost of 1.

We encountered a few problems while testing that needed to be accommodated for algorithmically. One problem was that our theoretical mapping from coordinate space to grid space, even with the top down camera view was slightly skewed. The grid space was tilted towards the left, which made it challenging for the program to correctly identify the coordinate space an obstacle was in in order to avoid it. For example, when setting up the obstacles, robot and goal in a straight line, it may in fact be that the robot is one grid column over according to the mapped space, and therefore would overshoot by trying to accommodate for the extra column shift. We accommodated for this by placing objects according to the camera view on the laptop to test specific path outcomes instead of placing items according to how they looked on the table.

Another obstacle we encountered was that the program assumes that with each move the robot moves one entire node, either forward, backwards, sideways or diagonally. Although theoretically this gives the optimal path the robot needs to follow to reach the goal successfully, experimentally we found that it was challenging to have our robot move one whole grid in any direction accurately and repeatedly. This was generally due to drag from the cables/motors and swifter range of motion in certain directions. We were able to accommodate for this error once we realized the problem by a combination of trial/error and analytical thinking. Aggravatingly though, this turned out to be a dynamic problem, for each time the robot was moved or picked up, drag seemed to be randomly created in different directions, thus it was challenging to fully accommodate appropriately with much precision in different trials.

Algorithmically, the path planning was highly accurate and repeatable. It successfully calculated the shortest path in all trials, when fed the correct information. The main source of error was in the actual implementation, where accuracy was hindered. The robot continued to have high predictability, as it did attempt to move along the designated path each time. Overall, the error between the calculated path and traversal path was high, but the repeatability of this error was also high, which leads me to believe that with more time to spend fine tuning the motor inputs and wheel design, accuracy could be improved.

Kinematics (Imtihan, Jayden, Laura)

The robot is holonomic, thus it has very simple kinematics. It has full range of motion in the 2D space, and could move horizontally, vertically, or diagonally. The wheel rotation needed to move one grid was 700 degrees in the horizontal and vertical direction, while in the diagonal was 990 degrees to move diagonally to the next grid. The movement was done using the kinematics equations we implemented for lab1, adapted to the specifications of this robot.

Analytical Results

Overall the robot reached its objective. The gridspace was defined by the smallest enclosing rectangle to the robot, and every successful test the robot reached its end goal, with 1, 2, and 3 obstacles. Object detection had some issues if the light in the lab room was set to be too bright, but in a dim environment, all the objects were properly detected in each of our 15 total runs (5 with each obstacles).

Conclusions

Overall the project taught valuable lessons in visual servoing, path planning, and robot design and iterative modelling. Our design achieved the basic objective, however with better vehicle design made with more stable materials, we could have explored more effective strategies for achieving these tasks.

Future Work:

Improvements could be made. For visual servoing, it is possible to implement a depth sensor and a camera module. Workarounds for getting openCV3 to work with a raspberry PI camera also exists, and that would allow a lightweight computer to be built into the robot. It would be interesting to see how the adapted D* Lite program works in a larger workspace and with dynamic object detection in place. It would also be helpful to compare different path planning algorithms, perhaps random based, to see whether mapped out routes are similar and to analyze the running time and error between them.

Video Demo link:

https://youtu.be/3-yCnFklS94

References

- https://en.wikipedia.org/wiki/A*_search_algorithm
- https://en.wikipedia.org/wiki/D*#D.2A_Lite
- http://idm-lab.org/bib/abstracts/papers/aij04.pdf
- http://images.slideplayer.com/17/3376199/slides/slide_25.jpg
- http://robotics.usc.edu/~geoff/cs599/D_Lite.pdf
- http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf
- https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf
- https://link.springer.com/content/pdf/10.1007/s12206-013-0725-3.pdf
- https://www.hindawi.com/journals/ijae/2016/5181097/
- https://link.springer.com/chapter/10.1007/978-3-319-31293-4_34
- http://robots.mit.edu/publications/papers/1998_07_Hac_Dub_Bid.pdf
- https://opencv.org/opencv-3-0.html
- https://en.wikipedia.org/wiki/Haar-like_feature
- https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html
- http://downloads.bbc.co.uk/rd/pubs/reports/1997-12.pdf
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.408.5710&rep=rep1&type=pdf