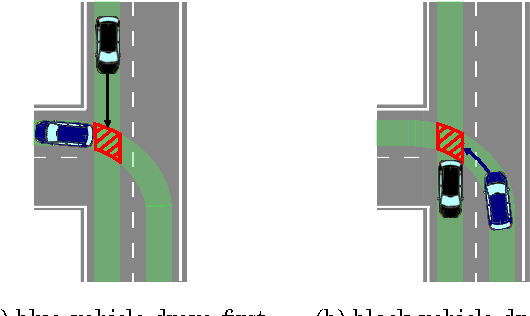
**CPSC 8810: Motion Planning Report**

**Adityaveer Raswan**

**Akshay Bhagat**



**Introduction to the Project:**

The Primary objective of the project is to make a Motion Planning Pipeline for an Autonomous Vehicle navigation and the test case is planning and handling an intersection with oncoming traffic and the traffic lights. A typical Motion Planning pipeline consists of a high-level global route planner which is responsible for guiding the vehicle in the right direction. Secondly, there must be a lower level route planner to avoid the obstacles locally. With the help of the sensors, the vehicle must be able to sense the static and dynamic obstacles in its path and plan the path accordingly to avoid those obstacles. Additionally, the vehicle has to review the present scenario and make decisions like Lane Change, Stop at Intersection, Yield to Traffic and pedestrians and so forth. This is done by a Behavioral Planner which carefully reviews the environment and makes the decisions to be taken by the vehicle.

**Project Details and Challenges**

In this project, we build a pipeline which demonstrates all these three key features in a simple scenario to understand the finer details and interactions between the three-layered Planning hierarchy. The Global Planning is done using Anytime - Astar algorithm for keeping the Global Planning simple yet fast enough to be reactive enough for the vehicle to reroute quickly. This global level planning is done using the information which is known by the vehicle through maps.

The Local Motion Planning is done based on the sensing horizon of the vehicle, which presents new information to the vehicle and thus demands a local re-planning based on the location of the obstacles. In order to keep the things simple and for integration purposes with the higher level planner, the lower level planning is also done using Anytime A-star algorithm, which again makes it reactive enough for the real-life driving scenarios because of the inflated heuristic. The heuristic then keeps on decreasing to optimize the path.

The newly detected obstacles are classified into either Static obstacles or dynamic obstacles based on the perception information. The static obstacles are added to the obstacle space of the map and the Astar is run again.

The dynamic obstacles present a unique challenge because of their dynamic nature, this problem is tackled in an innovative way by checking the collision between the dynamic obstacle path and the Car’s proposed path. If they are found to reach the point of intersection within some threshold time, then there is definitely a collision going to happen. The collision is checked at three points of the obstacle, the middle right, middle center and middle left points. Such a configuration ensures there are no collisions compared to checking at only one point at center (ignores edge collisions) or at two points at edges (head-on collisions would not be accounted for), also this is computationally inexpensive compared to checking at multiple points of the vehicle. Once a collision is detected the dynamic obstacle is propagated in time to the collision position and added to the obstacle space. After this, the anytime-Astar algorithm is run with the inflated heuristic which keeps on decreasing with time in search of the optimum solution if time permits.

Another major challenge was to depict the difference in a motion planning controller and the actual vehicle controller which are typically operating at different rates in a vehicle, meaning the motion planner might take its time to plan the algorithm, but the vehicle controller has to execute its commands at a fixed time step. This is shown by using multi-threading and queueing features in python which creates a new thread for the motion planner or the Astar algorithm. This means the vehicle can keep on running without waiting for the Motion planning to give a path. The anytime feature of the Astar makes it possible to get an initial sub-optimal path for the vehicle controller, and then keeps on improving the path with a reduced heuristic to get the optimal path.

There are three different circles, the first one depicts the outer sensing horizon for dynamic obstacles using a long-range sensor like RADAR. The second horizon is using shorter range sensors like LIDAR or CAMERA, which gives us the static obstacles information. The shortest inner circle is the Control Horizon beyond which the vehicle won’t be able to react in time if a path is changed, this represents the point beyond which getting the new path won’t make any difference, and hence the car takes the last available plan as the final plan for execution.

The behavioral planner is used to make decisions for different scenarios of traffic signs, namely Red Light, Green Arrow and Solid Green. In case of Red Light, the car stops at the stop sign by changing the goal to the stop position. For the Green arrow the vehicle has the right of way to turn left and its plans its path to reach the goal on the left, but this is also the case where an oncoming vehicle jumps the light and continues straight, so the vehicle has to account for the dynamic obstacle and plan an evasive maneuver using the local planner. In the final case, the vehicle is allowed to turn left but has to yield to the oncoming traffic. The vehicle only plans the path to go left when the other vehicle has crossed over, this is done by tracking the other vehicles position.

The code is written in python having a Node class to create various nodes, the A-star function definition is created along with calculating obstacle maps and calculating final paths. Additionally, collision detection functions were made which updated the obstacle map. The motion model of the vehicle is kept simple to up, left, right and down, but can be easily expanded to holonomic action constraints like going straight, left curve and right.

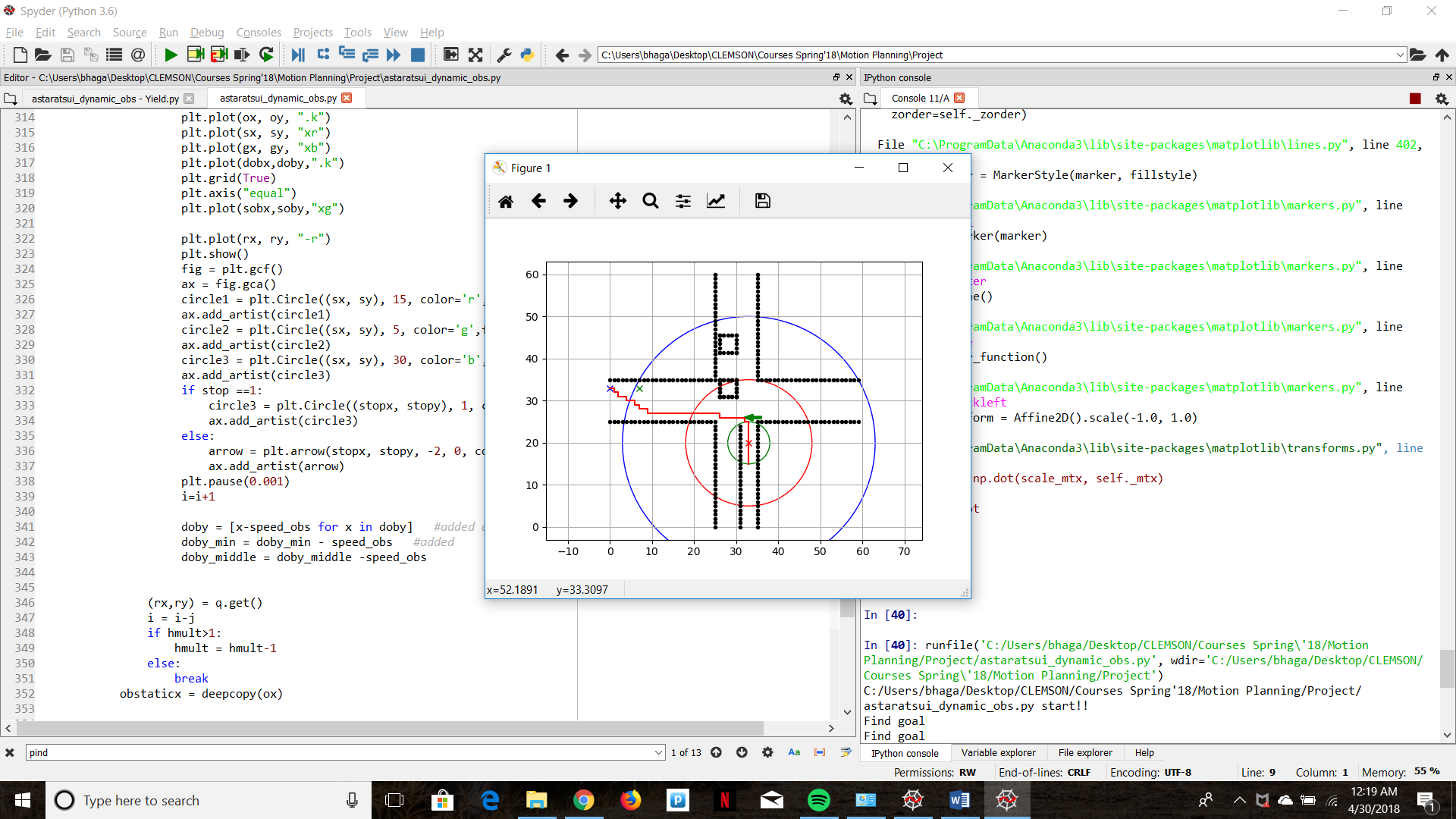
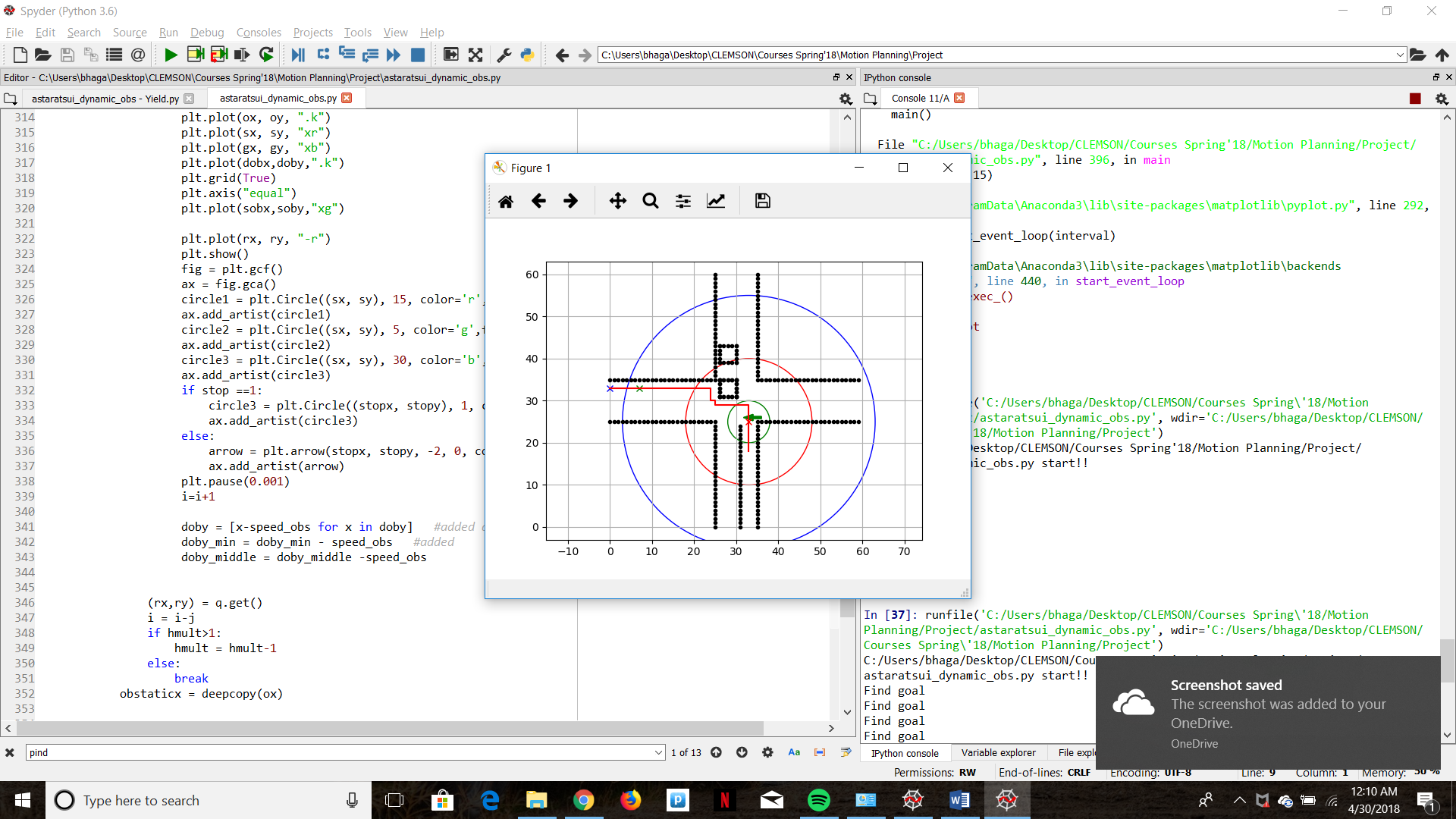


Fig2. Lower heuristic Optimized Anytime A-star for Dynamic Obstacle

Fig1. Higher heuristic Anytime A-star for Dynamic Obstacle

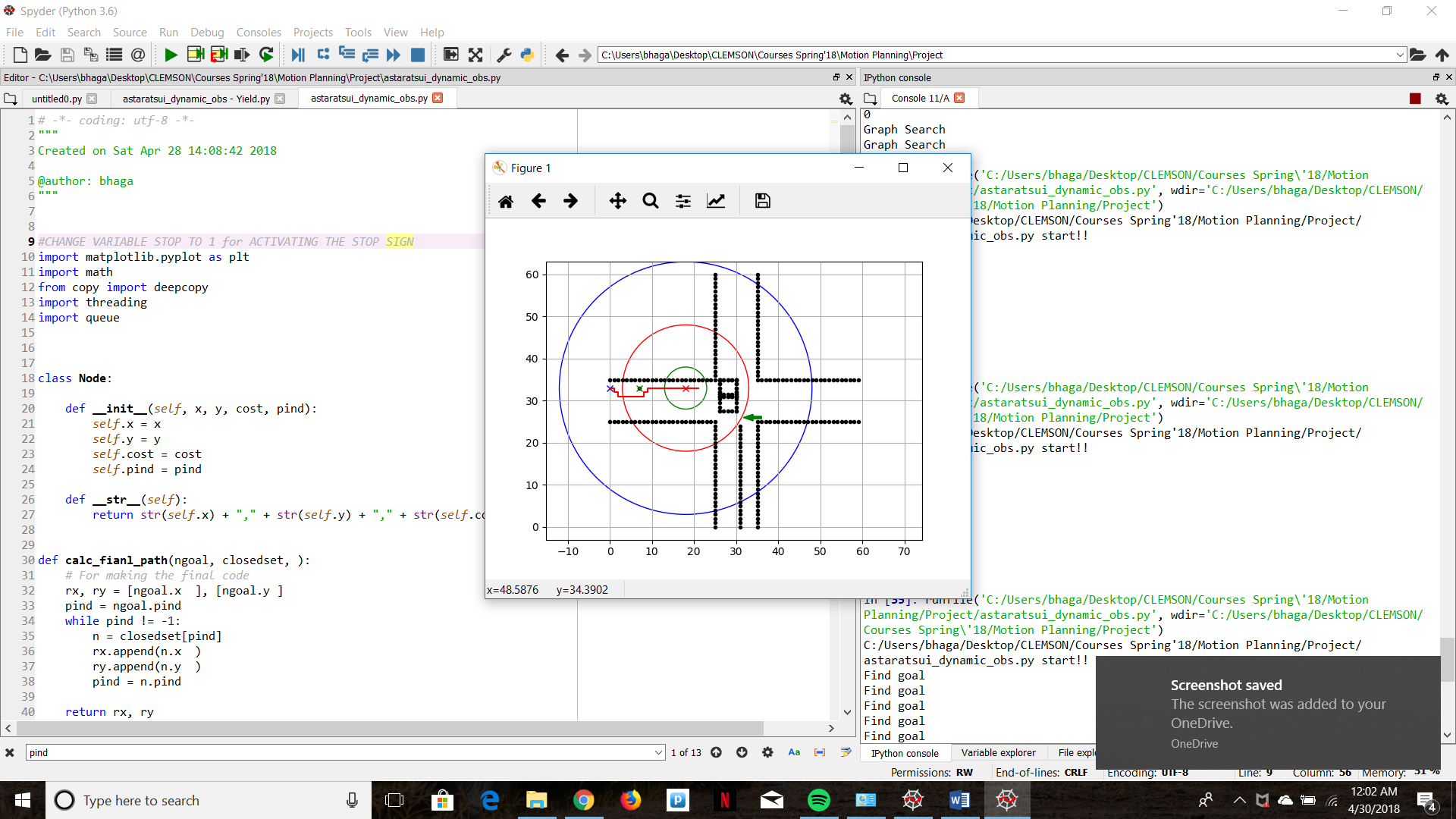
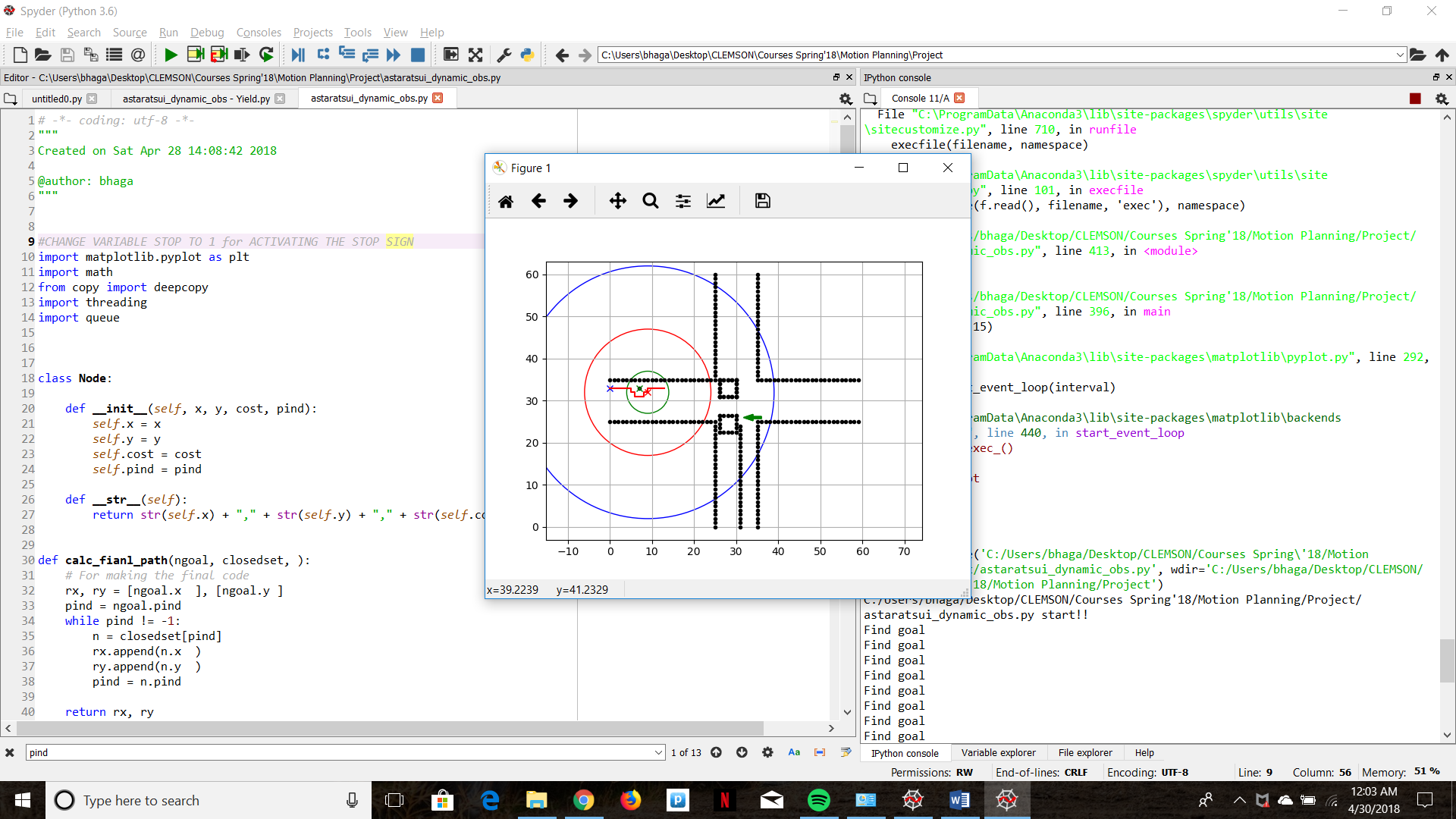


Fig3. Lower heuristic Optimized Anytime A-star for Static Obstacle

Fig3. Higher heuristic Anytime A-star for Static Obstacle

**Results**

The motion planning code is then visualized using matplotlib library for the three scenarios and the results are shown in the form of animation below, and also the two files shared which are StopandNormal.py and Yield.py. In the StopandNormal.py file the scenario changes from stop to left arrow by making stop variable from 1 to 0.

1. In the first scenario, as expected the vehicle stops at the sign and waits for the signal to become green.
2. In the second scenario, the vehicle plans to go left and then finds a vehicle jumping the light on the other side and accounts for it by dynamically planning in the local level planner.
3. In the third scenario, the vehicle waits for the other vehicle to pass and then plans to the path to the left

**Future Work**

The future work on the topic includes:

1. Expanding the anytime Astar algorithm to Dstar lite such that the dynamic planning becomes even more optimized for the dynamically changing situation.
2. The vehicle motion primitives have to be changed from up, down, left and right to straight, left turn and a right turn for making it more like a nonholonomic car-like agent.
3. The car speed can also be taken as a state or action to be adjusted, which will give us more degree of freedom for obstacle avoidance
4. The global and local planning algorithms can be made more generalized to take into account more scenarios.
5. The behavioral planner can be expanded to include more functions like vehicle turn signals, vehicle intent consideration, pedestrian tracking, etc.

**References**

1. Likhachev, Maxim, Geoffrey J. Gordon, and Sebastian Thrun. "ARA\*: Anytime A\* with provable bounds on sub-optimality." Advances in neural information processing systems. 2004.
2. Wilkie, David, Jur Van Den Berg, and Dinesh Manocha. "Generalized velocity obstacles." Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on. IEEE, 2009.
3. AtsushiSakai. “AtsushiSakai/PythonRobotics.” GitHub, github.com/AtsushiSakai/PythonRobotics.
4. Koenig, Sven, and Maxim Likhachev. "D^\* Lite." AAAI/IAAI 28 (2002).
5. LaValle, Steven M., and James J. Kuffner Jr. "Rapidly-exploring random trees: Progress and prospects." (2000).