The Multipurpose Autonomous Agent Project: Experiential Learning for Engineering Assistive Artificial Intelligence

Chad Mello, James Maher and Troy Weingart
Department of Computer & Cyber Sciences, United States Air Force Academy, Colorado Springs, CO 80840, U.S.A.

Keywords: Experiential Learning, Assistive Technologies, Artificial Intelligence, Machine Learning, STEM Education, Engineering, Teamwork.

Abstract: Assistive Technologies (AT) and Artificial Intelligence (AI) that support humans in decision making and in difficult or dangerous tasks are in high demand. We created a two-semester capstone project, for undergraduate seniors, providing the opportunity to build an assistive AI algorithm implemented on a skid-steer rover platform. By the end of the program, students created a system with the potential for assisting humans in dangerous indoor situations such as: gas leaks, bomb threats, fires, and active shooters. Our unique approach allowed the skid-steer rovers to autonomously navigate indoor areas never before encountered or previously mapped. Students used deep behavioral cloning techniques coupled with deep reinforcement learning to train the rovers for speed, steering control, and cornering. Outfitted with nothing more than a depth-sensing optical camera, an inexpensive autopilot, and an onboard, assistive NVIDIA Jetson Xavier NX computer, the rover quickly scanned and oriented to a new environment and then located objects of interest. The students' final product demonstrated impressive abilities and skills demanded by industry in developing AT and AI platforms for mission-critical applications. Herein we share our approach, technology stack, experiences, and artifacts produced by our students at the end of the project.

1 INTRODUCTION

Post-secondary Computer Science (CS) departments are tasked with educating and preparing graduates to fill modern, evolving CS related jobs. Many CS graduates are highly knowledgeable; however, (McGunagle and Zizka, 2020) found that employers desire problem-solving graduates who are able to work harmoniously as part of a team. CS undergraduates rely on their degrees as a foundation for beginning a career in industry; however, even the best students experience a skills-gap when they work their first job. CS faculty members identified a desire to provide more projects working real-world, industry problems to address this skill-gap (Valstar et al., 2020). While there is little difference in *academic* performance between students who intern and those who do not, graduates without industry experience often find themselves less employable because they lack practical experience, good technical and interpersonal skills, and the ability to work effectively in teams (Kapoor and Gardner-McCune, 2020). Recent studies, involving 536 multi-institutional CS students, found that only 57.5% of undergraduate CS students completed an internship prior to graduating (Smith and Green, 2021; Kapoor and Gardner-McCune, 2020).

Machine Learning (ML) shows potential for solving problems in many areas of science, medicine, and engineering (Farjo and Sengupta, 2021; Lürig et al., 2021; Azari et al., 2020; Rutherford, 2020; von Lilienfeld and Burke, 2020; Akbilgic and Davis, 2019; Toole et al., 2019; Fraley and Cannady, 2017; Trister et al., 2017); therefore, it is important to have a portion of CS curriculum in higher education devoted to ML. Yet, according to (Shapiro et al., 2018), educational offerings in CS departments do not reflect this reality. This paper presents a practical project for developing teamwork and problem-solving skills that are in high demand from industry (McGunagle and Zizka, 2020).

Recently, we designed an applied, Deep Learning (DL) capstone project to improve students' knowledge working with Imitation Learning (IL) and Reinforcement Learning (RL) in an open-world environment. The *Multipurpose Autonomous Agent Project* (MAAP) is a year-long undergraduate program offered to senior CS majors; it targets skills and insights that are valuable to industry as well as the United

Mello, C., Maher, J. and Weingart, T.
The Multipurpose Autonomous Agent Project: Experiential Learning for Engineering Assistive Artificial Intelligence.
DOI: 10.5220/0011715500003470
In *Proceedings of the 15th International Conference on Computer Supported Education (CSEDU 2023) - Volume 2*, pages 252-263
ISBN: 978-989-758-641-5; ISSN: 2184-5026

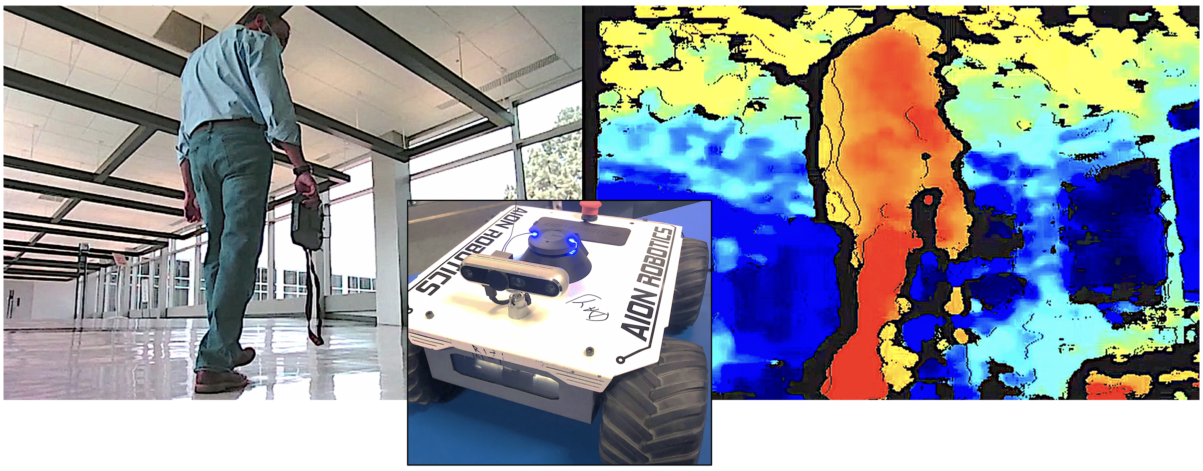


Figure 1: The MAAP rover (center) in action. The left and right images are a snapshot of simultaneous streams of data that the rover collects through its front-mounted camera and processes in real time. The left image represents a RGB stream while the right side represents rasterized 3D distance information of the same scene. This data is pre-processed, stacked, and then fed to the rover's onboard neural networks. The rover then bases its actions on the policy network's throttle and steering mixture outputs as well as its object recognition outputs.

States Air Force Academy (USAFA) at the time of this writing: Assistive Technologies (AT), Artificial Intelligence (AI), DL, IL, RL, and Computer Vision (CV).

MAAP offers students direct experience by tasking them with designing, programming, training, testing, and deploying AT under the supervision of an experienced mentor. The project holds inherent challenges that a small team of students must identify and overcome to be successful. The final product combines hardware, software, and advanced algorithms into a cohesive, ready-to-use package. The result is an autonomous, skid-steer rover built using Modifiable Commercial off the Shelf (MOTS) parts and open-source Application Programming Interfaces (APIs).

The MAAP rover may be deployed to an indoor setting that has not been mapped or is otherwise previously known by the device. The rover is equipped with a front-facing camera and a distance-sensing device, giving it the ability to traverse and explore an environment by navigating hallways and rooms while simultaneously scanning for an objective object. The objective can be detected through infrared, gas, or video sensors installed on the rover platform. Video and data may be streamed in real-time to a console app running on a tablet or laptop for real-time feedback. The final project deliverable is a generic package ready for further use in other object detection scenarios in unmapped environments.

This paper makes three contributions: (1) we present a Project Based Learning (PBL) framework for use in a team-based undergraduate capstone, (2) we identify the resources needed to replicate and/or improve MAAP, and (3) we share our assessment of the MAAP project's effectiveness. The remainder of this paper is structured as follows: Section 2 summarizes related works to this project, Section 3 provides a description of the hardware and software platform, Section 4 describes the course format, Section 5 describes student progression through the year-long course, and Section 6 describes our conclusions and proposed future work.

2 RELATED WORKS

MAAP provides CS majors with experience in applied ML, complex problem solving, and harmonious teamwork. MAAP engages students with an interesting set of problems that cannot be solved only with the knowledge they possess at the beginning of the program. Teaching students to make design decisions for ML implementation is more difficult than teaching core ML concepts (Sulmont et al., 2019), and MAAP focuses on experiential learning beyond core ML concepts. Coincidentally, just as students learn complex skills through observing expert demonstration, MAAP focuses on a similar concept for teaching machines complex tasks and functions through Imitation Learning (IL). This section provides a background on IL and current teaching methods.

2.1 Imitation Learning

In environments where complex autonomous functions are required, it is easier to teach desired behavior through demonstration rather than attempting to engineer it (Osa et al., 2018). IL reduces the problem of teaching a task down to a human providing task demonstrations and then recording how the human performed this task. Equipping an agent with modern sensors gives it the ability to collect large amounts of data to rapidly process, learn and create maps that transform data into actions (Hussein et al., 2017a). IL encompasses a group of algorithms designed to learn from and mimic the behavior of humans or animals exhibited under specific circumstances; these algorithms discover a mapping between observations and actions via a learning process (Hussein et al., 2017b).

Behavioral Cloning (BC) is a subset of IL popularized by two papers published by NVIDIA that utilize a Convolutional Neural Network (CNN) to go beyond basic pattern recognition and learn the entire processing pipeline needed to perform an action. NVIDIA researchers published an initial paper (Bojarski et al., 2016) and a followup paper (Bojarski et al., 2017) that inspired the creation of an online Udacity project centered around a training car to drive itself in a simulator (Dominique Luna, 2021). Figure 2 shows the trained model driving the car in the Udacity simulator.

2.2 A Simple Introduction via Udacity

The Udacity demonstration simulates a car with rack & pinion steering traversing a consistent and clearly marked race track. The simulator collects and processes visual data related to steering input, provided by a human driver, as the car is driven around the track. A CNN is built to accept individual, pre-processed video frames, generated by the simulator's scenery, as input and then predicts the steering angle as output. When training the CNN, the steering angle is provided as labeled output and the sensor collected imagery data provides the inputs.

The Udacity simulator introduces students to the concept of an autonomous pipeline, i.e., sensor inputs are utilized to train a model and generate a cyber-physical output; however, the Udacity simulator cannot perform outside of a very specific environment. The simulator provides students an understanding of the problem and an elementary approach to solve it. The autonomous steering demonstration in a simulator cannot address non-determinism in the physical world. In addition, the CNN trained in the Udacity simulator lacks the ability to predict speed control and object avoidance, i.e., the learned policy will not translate from the simulator to the real-world.

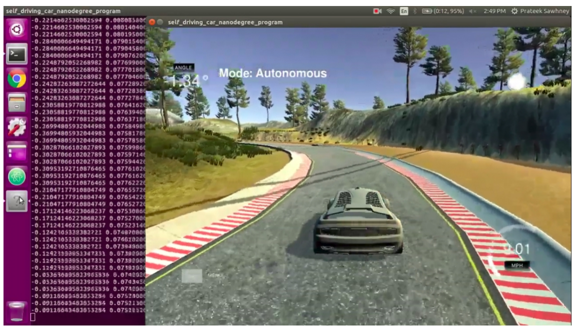


Figure 2: Car driven by trained CNN through Track 1 in the Udacity simulator.

3 MAAP OVERVIEW

The MAAP capstone is a 6-credit hour program. The program centers on integrating ML with autonomous rovers to provide applied solutions to real-world, open set problems.

3.1 Product Requirements

The MAAP product students are required to deliver at the end of the program is a physical skid-steer rover capable of traversing indoor environments with autonomous steering and throttle control. The rover will find its way out of dead ends, smoothly turn corners, minimize contact with walls and stationary objects, tread carefully around people and pedestrians, accelerate smoothly when there are no objects in its line of sight (conversely slow its speed according to its proximity to objects in front of it), avoid moving objects, turn away from or even reverse itself to avoid oncoming objects, and perform scanning maneuvers (360 degree, in-place turns to scan for objectives). In addition, the platform utilizes computer vision and object recognition using mounted sensors to execute searches for specific objectives defined in the course. The rover performs these actions in the real-world, avoiding the limitations that result from deterministic learning environments. Ideally the final product would utilize some form of spatial mapping, e.g., Visual Simultaneous Location and Mapping (VSLAM) to avoid exploring areas previously searched; however, due to the limited student time, VSLAM techniques were not covered or required.

3.2 Hardware Choices

The students were not involved in hardware decisions; hardware was supplied at the beginning of the capstone project. We took into account budget, safety concerns, and feasibility when considering the hardware choices for this program. Smaller 1/10 scale RC cars are typically used for similar projects and are overall less expensive. We desired the maneuverability of larger four-wheel drive skid-steer rovers as well as the simpler and more rugged build. We purchased all-metal, fully-assembled, mid-sized four-wheel skid-steer platforms equipped with Pixhawk Cube Orange (or Black) autopilots (Ardupilot, 2022). A number of reasons went into our decision for purchasing these machines:

- We wanted to afford students the opportunity to train and predict ML and AI models with the same hardware which rides on the rover without transferring trained models from a more powerful workstation. The autopilot, NVIDIA GPUs, 1 TB of hard-drive data storage, cameras, distance finders, etc. are all onboard the rover. Conducting all ML and AI model work on the same hardware provided a more efficient work environment as students improved their models.
- We desired to have all AI and ML algorithms to train and predict on the rover, without the need for a remote computer connection. This significantly reduces latency with data streaming, image processing, and command execution. RC transceivers were on hand at all times to override rover functions in cases where code might create an undesirable situation.
- We wanted to lessen situations where rovers could become stuck when presented with rough, bumpy or cluttered terrain. We chose large tires and independent electric motors with adequate torque so that rovers would be capable of rolling over rocks, boards, nails, and other debris that could be present.
- Our rover units are capable of carrying larger sensors, computers and onboard batteries. In future renditions of MAAP, we plan to add more sensors and other equipment to these devices. These platforms are more accommodating *via* their larger footprint and payload capacity.

We purchased a long-range *PowerBox Systems* Radio Core RC System (PowerBox Systems, 2022) for controlling the rovers during data collection. In addition, each rover is equipped with an *Intel® RealSense*™ D455 depth-sensing camera (Intel, 2021), the NVIDIA® *Jetson Xavier*™ *NX* for accelerated AI execution (NVIDIA, 2022). The NVIDIA® *Jetson Xavier*™ *NX* contains a 6-core ARM CPU, 384 GPU cores, 8 GB RAM, and a 1 TB SSD. See Figures 3 and 4 for profile and internal details.

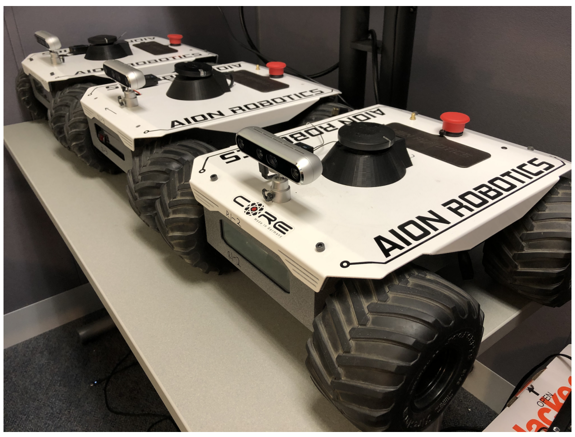


Figure 3: Three rovers used in the MAAP capstone. In addition to the standard RC radio, autopilot, and GPS systems, we attached an *Intel® RealSense*™ D455 depth-sensing camera.

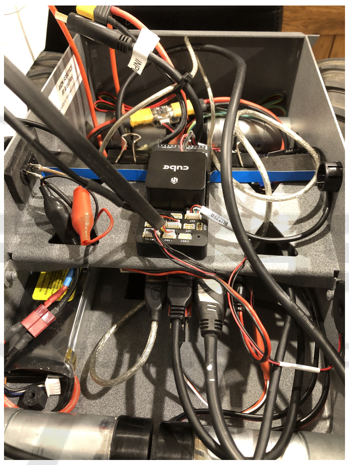


Figure 4: The MAAP rover internals include an independent assistive computer located under the autopilot cube.

3.3 Software and Tools

We relied heavily on cross-platform software for MAAP. This was done out of necessity. All of the students at our institution are issued Windows laptops during their Freshman year. Consequently, we wanted to ensure that the software solutions would run on their systems as well as with our hardware–much of which utilizes Linux-based operating systems.

Students are required to use several pieces of free, cross-platform software while developing code for the rovers. Python, i.e., the *Python 3.8* interpreter (Sanner et al., 1999), was chosen for the programming language. PyCharm Community Edition (JetBrains, 2022) was chosen for the Python development environment. PyCharm Community Edition is a free, robust development environment that offers easy virtual environment creation, debugging and syntax checking. To communicate with drone hardware via the *MavLink* protocol, we utilize two drone-related Python libraries: (1) Pymavlink and (2) DroneKit. Machine learning and computer vision are facilitated by *TensorFlow*, *Keras*, and *Open-CV*. Finally, we include the *PyRealSense* Python library for interfacing with the *Intel® RealSense*™ D455 depth-sensing camera. All of the Python packages listed in this paragraph can be downloaded from the PyPi package repository (Python Software Foundation, 2022) and installed in the students' Python environment using the *pip* command. Note that there was no need or requirement for simulator software or Software in the Loop (SITL). All work can be performed on the actual hardware in the physical environment.

4 PROGRAM PROGRESSION

The MAAP capstone course was organized into four-week sprints. Class time for capstone work averages 5 hours per week with an instructor present and an additional 8 hours of non-instructor work time per week. At the start of each sprint, the capstone team and instructor plan the sprint by considering what should be accomplished in the four week period. Scrum meetings occur at the beginning each week and last 30-45 minutes. The work environment is designated to encourage open ideas and rapid prototyping.

Periodic gate checks are conducted by the instructor to ensure work is progressing in a timely fashion. The instructor may conduct hands-on workshops to help students apply theoretical knowledge to the physical hardware environment. To form a complete picture of an end-to-end solution, each major sprint concludes with a demonstration where students must present a working prototype. Twice during the semester the students must present their work to selected faculty members outside of the project and the team is graded by an uninvolved reviewer.

MAAP combines several complex topics, and the instructor provides the essentials necessary to complete sprint tasks. It is important to consider the students' ability to learn and implement new material, without overwhelming the students in the four-week sprint period. For example, if it becomes obvious that the team is struggling with identifying, evaluating, and narrowing potential CNN models for use in policy learning, the instructor may guide the team towards the models that are likely to bring the biggest benefit to the project. An important outcome for this course is to produce graduates who have both knowledge and implementation skills, yet the instructor must maintain cognizance of the project's direction and guide the team away from ideas that could derail it.

4.1 Provided References

We chose to provide trade books and current papers as a reference to the students. The advantages to using trade books over traditional textbooks are: (1) quality publications provide readers with robust theory while filtering all but the essentials of how to implement the theory, (2) trade books are often considered more readable, making examples and instruction more accessible to undergraduate students, and (3) students who go on to become professionals are likely to turn to trade books when working in industry (Schultz, 2014), (Smolkin et al., 2013). We chose several texts for the course:

- Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow 2nd Edition (Géron, 2019)
  A Géron - 2019; ISBN-13: 978-1492032649
- Deep Reinforcement Learning Hands-On:... (Lapan, 2020)
  M Lapan - 2020; 978-1838826994
- Reinforcement Learning: Industrial Applications of Intelligent Agents (Winder, 2020)
  P Winder - 2020; 978-1098114831
- Hands-On Computer Vision with TensorFlow 2: Leverage deep learning to create powerful image processing apps with TensorFlow 2.0 and Keras (Planch, 2019)
  B Planch - 2019; 978-1788830645

The Papers We Chose for the Students to Review Were:

- End to end learning for self-driving cars (Bojarski et al., 2016)
- Explaining how a deep neural network trained with end-to-end learning steers a car (Bojarski et al., 2017)
- Imitation Learning: A Survey of Learning Methods (Hussein et al., 2017b)
- An Algorithmic Perspective on Imitation Learning (Osa et al., 2018)
- Evaluation of pre-training methods for deep reinforcement learning (Larsson, 2018)
- Learning to drive by imitation: An overview of deep behavior cloning methods (Ly and Akhloufi, 2020)
- Deep reinforcement learning for autonomous driving: A survey (Kiran et al., 2021)
- Mobile robots exploration through CNN-based reinforcement learning (Tai and Liu, 2016)

- Recent advances in imitation learning from observation (Torabi et al., 2019)

4.2 Bootstrapping: Code and Libraries

MAAP requires students to write a significant amount of Python code. We provide starter code as well as Python virtual environment setup instructions to provide a starting point. The following are the items we provide to the students:

- Custom *Python libraries* containing common functions for communicating with our rovers, along with commonly-used commands and processes,
- *skeleton code* that provides a working Main function and stubs that students are required to complete,
- coding standards and naming conventions, and
- an existing Github repository where are code and documentation are stored

These items provide structure, allowing students to quickly focus on the major tasks at hand. Students have an opportunity to explore the initial project code as well as some functional examples that they may modify to fit their needs.

4.3 Mini Lectures & Labs

The instructor provides theory and industry experience for the team through additional labs. During the program, we provided several lessons that are designed to closely mimic a given sprint's upcoming tasks (see section 5 for breakdown of the labs we provided to the students). We cover topics such as DroneKit APIs, Open-CV, image processing, TensorFlow, Keras, data collection, data preparation, deep CNN models, deep RL, model performance analytics, and model fine-tuning techniques. Most lectures include some form of sample code in Jupyter Notebooks (Kluyver et al., 2016) that students may use as starter code for their tasks.

4.4 Team Assessment

At the end of each sprint, students are graded as a single unit. The instructor reviews and assesses sprint artifacts, demonstrations, and assessments from outside faculty reviewers. The artifacts, demos, and outside assessments are weighted and combined to yield a final grade. The team is encouraged to be bold and imaginative; however, members are also responsible for recognizing when they are at an impasse and when they should consult their instructor. The instructor allows the students to wrestle with team conflicts, bad ideas and some failures, but keeps a close eye out for when those things begin to compromise the overall success of the project. Typically, by the end of the first two sprints, the team will have learned each member's strengths and weaknesses and will begin to demonstrate improved efficiency as a result.

5 SPRINT BREAKDOWN

The entire project lasts for eight, four-week sprints. It is critical that the team make progress during every sprint in order to meet the requirements by the end of the program. In the following sections we outline the major events, instructor interventions, and results from the sprints at our institute.

5.1 Sprint One: Orientation

The team was introduced to the project, its requirements, high level concepts, source repository, development tools, and reading material (see section 4.1). The main objective is for each student to learn and understand IL theory. Students already had exposure to AI and ML as well as RL, and this prior knowledge served as a base upon which to build their understand of IL and behavioral cloning. Artifacts from this sprint consisted of a report that summarizes the team's understanding of the project requirements, toy code similar to the Udacity demonstrations (discussed in 2.2) to demonstrate an understanding of a simplified example of behavioral cloning by each team member.

5.2 Sprint Two: Data Collection Utility and CNN Cloning Design

The team learned the essentials of data collection from the previous sprint. For sprint two, they created a program that collects data from the rover hardware stack. This includes writing code that interfaces with the rover's autopilot and the RealSense camera streams. Steering and throttle data from the rover was captured and matched with imagery and depth data from the Intel Realsense camera. This data was stored and then later used to train a CNN model. The team was free to use the starter code we provide to interface with the camera sensors and the rover's autopilot (see section 4.2).

The data collection program (i.e., the recorder) works by waiting until the rover is armed via the remote RC handset. Once armed, the program begins data collection. The human operator may collect as little or as much data as desired. Human operated driving sessions lasted anywhere from 1 to 15 minutes. When the rover was disarmed via the RC handset, the dataset was closed. A dataset consists of a video file that contained both RGB and distance measurement streams, a data file, and a log file. The data file temporally matched steering, throttle, ground speed, and heading information with each frame in the video file. Note that only steering and throttle are currently used in MAAP, leaving the other variables available for future use with more complex models. The log file recorded autopilot status, recording time, sensor status, and any exceptions that might occur during the session.

Simultaneously, the team began training the initial CNN model for behavioral cloning. The instructor directed the team to begin with a modified version of the CNN model introduced in (Bojarski et al., 2017). The team built this model using Keras with TensorFlow as the back end. In this project, the team created a more complex model than what is presented in the Bojarski paper. The key differences are as follows:

- **Input:** the original model accepts three frames from center, right, and left cameras at the same point in time, with steering angle as a label during training. Our model now accepts a single RGB frame from a centered camera stacked with pixel-aligned gray-scale depth sensor data, using linear throttle and steering numbers as labels. The resultant CNN is now a multi-label model that takes throttle into account as well as steering.
- **Output:** the original model renders a single value representing steering angle, while our modified regression model now provides predictions for both steering and throttle control.

Artifacts from this sprint included (1) a utility program to collect training data, from the rover, for use in training the CNN cloning model, and (2) an initial CNN ready to train with data. The data collection and model training started in the following sprint.

5.3 Sprint Three: Training Data and CNN Training and Experimentation

The team began data collection, utilizing the utility program developed in Sprint 2, as well as trained their initial CNN model. Data collection was time consuming. We chose to collect driving data from two of six floors in a large building, reserving the remaining floors for performance evaluation. To introduce diversity into the training data, each student on the team was tasked with collecting a total of two hours of driving data during this sprint. Students used the utility program to record driving maneuvers as they manually guide the rover around the building.

5.3.1 Driving Sessions

Several types of driving sessions were collected at various times of the day to account for natural and artificial light, light glare, artificial and overhead lights as well as low-light conditions. We identified four categories of driving sessions for which we collected data:

1. Smooth forward driving in empty, clear hallways around the building in clock-wise and counter-clockwise fashion. The rover was kept more or less to the right side as it traversed the hallways and around corners. Some sessions had the throttle locked between a low and high setting. Throttle and steering are varied, but driving was performed with relatively smooth steering and throttle input, i.e., no erratic maneuvers. Throttle was generally increased when driving in a straight line with no objects in sight; throttle was lowered when coming close to a wall or corner at the end of a hallway.
2. Object avoidance sessions that involved driving around various inanimate/static objects and obstacles that are set in random areas throughout the hallways. Obstacles might be tables, chairs, boxes, trashcans, maintenance equipment, doors that open out into the hallways, litter, and stationary people who are either standing or sitting at different angles. The path of least resistance is taken, but in the case where an object might be in the center of the hallway, the drivers were asked to vary left and right turns when avoiding those objects so as to avoid introducing a right-hand or left-hand bias. The drivers were also asked to vary when they begin maneuvering to avoid an object. In some instances a driver began maneuvering well ahead of time while in other instances the driver maneuvered at the last moment to avoid an object. Throttle varied, but it always decreased when approaching an object.
3. Object avoidance sessions involving moving objects such as people, carts, trash cans, and rolling chairs. Throttle and steering were utilized to avoid colliding with moving objects. In the case where a person or object is moving towards the rover, the driver would decrease throttle and change course. If the object also corrects and continues to approach the rover, the driver would come to a stop or even reverse the throttle to begin a backwards movement. Since there were no rear sensors,

backup throttle was only used for a short duration while the driver attempted to take the first forward path avoiding the object altogether.

4. Entering adjacent rooms where entrance is feasible. Drivers turned into the closest doorway where the door was open and the path was clear. Throttle was reduced considerably as the rover was driven past the threshold and further into the room. Gradually the throttle increased according to the condition of the room (i.e. how much clutter is present). Rovers were driven around and under desks and chairs and around any other obstacles present in the room. Once the room was traversed, the driver maneuvered the rover back into the hallway.

All sessions were stored and labeled according to the driving session types mentioned above. All original log and label files were retained along with the full-resolution sensor and video streaming in rosbag format (a Robot Operating System (ROS) file format for storing ROS message data).

5.3.2 CNN Training

Training began before the entire data collection process was complete. Small subsets of driving session type 1 data (see Section 5.3.1) was used to begin initial CNN training near the beginning of Sprint Three. The training data was further prepared by creating and running a special utility program that rips individual frames from the video streams, resamples and reprocessing the images while also labeling the frames with steering and throttle numbers from the matching data file. In addition, an inline data generator was also created to wrap the raw data. The data generator was used to randomize and split data into training, validation, and test splits as well as to feed mini batches to the model upon request as it trains.

The training scripts saved the model only if the validation loss improved over the previous epoch. Initial training began over a maximum of 50 epochs, with performance flattening between 17 and 20 epochs. The training script was designed to resume training as new data arrives by de-serializing previously trained best model.

Finally, a control program was constructed to read camera feeds in real-time while feeding them into the newly trained CNN. The raw CNN output was de-normalized and passed directly into the autopilot channel feeds for throttle and steering, overriding normal function. This program ran in a continuous loop at 15 frames per second, constantly plugging in the cloning model's actions to the rover's autopilot.

Artifacts included a demonstration of a rover that can somewhat navigate throughout a single hallway (but not very well). Even though the rover could navigate simple hallways and around corners 30%-50% of the time, it is an exciting time as the team observes the rover "magically" roaming the hallways and largely avoiding walls and objects.

5.4 Sprint Four: Model Experimentation and Shortcomings

The team was guided towards the discovery of several major improvements to the training process. First, the numerical range for both steering and throttle is between 1000 and 2000 with 1500 representing the "center" or neutral. Anything less than 1500 represents a value towards a left-hand turn and anything over 1500 represents a value towards a right-hand turn when for steering input. Similarly, for throttle inputs, anything under 1500 represents backwards movement and anything over 1500 represents forward motion. Normalizing these variables using a min–max normalization limits the range between 0 and 1. This helped with decreasing the time it took for the CNN to converge during training. Furthermore, the learning curves were smoother and showed less "bouncing" than before applying normalization. In addition, the team discovered that training using small, randomized frame sequences of 13 frames per sequence markedly improved overall performance; the rover's steering and throttle stability improved along with its ability to smoothly round corners. Another discovery of note: experimentation provided evidence that randomized sequences using a CNN may yield better performance over more complex Long Short Term Memory (LSTM) models.

The team experimented with various deep models until they discovered models that demonstrated potential for good performance. The artifacts for this sprint were: (1) a discussion of model performance and (2) demonstrated improvement using a behavioral clone model. The rover was better able to negotiate corners, showing marked all-around improvements. In fact, the rover was able get out of most dead end situations, navigate around most objects, and even slow down or stop around people, mimicking extra caution around people and pedestrians. The performance was impressive as well as inspiring to the students. However, the team soon discovered that a blind cloning effort can only take performance so far before weakness begins to show.

5.5 Sprint Five: Improved Performance and Need for Reinforcement Learning

Despite the impressive performance gains in the previous sprint, the team began to understand that a single CNN cloning model simply could not provide the functionality MAAP requires. This is due to model's overly simple action (i.e., output) policy. The model was only capable of learning a distribution of actions over a finite training set. The model had no way (except through inherent bias in the training data) of learning when it was better to turn left vs turning right, or when it is more beneficial to back up vs doing an in-place 360 (remember, this is a skid steer rover). This is where behavioral cloning alone begins to show its limitations.

The instructor discussed concepts related to transfer learning (domain adaptation) coupled with reinforcement learning models best suited for learning policies over a continuous action space. We pointed the team towards Actor-Critic models, specifically Advantage Actor-Critic (A2C), coupling it with the now much-improved, pre-trained Bojarski CNN model the students had been developing up to this point. This opened the door to a host of potential performance gains. The idea came from several sources we provided as reference material for this project: *Deep Reinforcement Learning Hands-On*; chapters 12 and 17 (Lapan, 2020), *Reinforcement Learning: Industrial Applications of Intelligent Agents*; chapters 7 and 8 (Winder, 2020), and the paper *Evaluation of pre-training methods for deep reinforcement learning* (Larsson, 2018).

This sprint marked the most critical point in the project, because it required a deeper understanding of RL and IL as well as advanced skills with TensorFlow. The instructor closely monitored and advised the team through the initial approach and programming efforts. The resulting model was an A2C architecture where the actor and critic shared the same modified Bojarski CNN. The discounted rewards were based on a simple calculation involving the distance of objects in the field of view. A 2D Gaussian matrix was applied to the depth sensor pixel values from the camera, creating a gradient scoring system where pixels of objects in the periphery were given a better score over objects in the forward center of view (objects in the center of view were actually penalized more). This scoring system was designed to teach the model to always look for the path of least resistance (i.e. facing where fewer objects are in the way). This created a policy whereby the model favors actions that result in the clearest path ahead (see Figure 5). The maximum movement was restricted so that the rover would not be turned completely around to avoid taking a valid path ahead of it in favor of a better path behind it (i.e. a path it most likely already traversed). The artifacts were the related Python code that contained the new A2C model and the code required to train, test, and analyze the new model.

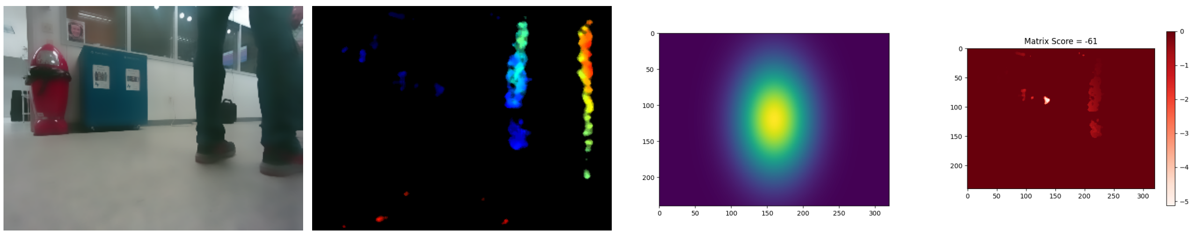
5.6 Sprint Six: Continued RL Work

Eventually a model emerged that would normally require thousands of training sessions using a virtual environment before it would be able to do anything useful. However, the team came up with a way to pre-train the new A2C model using training data that was collected during Sprint Two. The training was not optimal because the model was not allowed to explore during these training sessions; however, the training was adequate for a bootstrap where transfer learning techniques could be used to optimize the model over additional incremental training sessions. An additional Python script was created to facilitate the rapid incremental training of the model. The program waited until the rover was armed before starting a new training session, and training required a human user to operate the rover. Once the training session began, the rover would begin traversing the interior from the location from where it was armed, using its new A2C model. Anytime the rover makes contact with a wall or object the user simply flips the RC hand control switch to disarm the rover, thus ending the session. The human user could then direct the program to immediately start training a new session by flipping a switch on the RC controller. This vastly improved the speed of data collection.

The results were encouraging. The rover was able to traverse busy hallways and rooms right out of the box in areas the model had never visited or trained over beforehand. Incremental training was time consuming, but incremental improvements took relatively little time to notice. We did not have a virtual environment to train the model over thousands or even millions of sessions. This remains an open agenda item for future iterations of MAAP.

5.7 Sprints Seven & Eight: Mission Deployment

The A2C model could use more tweaking and training; however, there was no more time available to continue that effort. By this time we addressed the project's object recognition objectives by implementing bolt-on object recognition models. The team settled on a pre-trained object recognition model



(a) Original image (b) Heat map of original image (c) 2D Gaussian matrix (d) Result of inverse scaling b with c

Figure 5: The reward system for our A2C model consists of penalizing the closest objects as they approach the forward center of view, subtracting those points from the maximum points in the image. This is accomplished by applying an inverse gradient scale using a multivariate (2D) normal distribution against the distance value for each pixel as measure by the distance sensor in our camera.

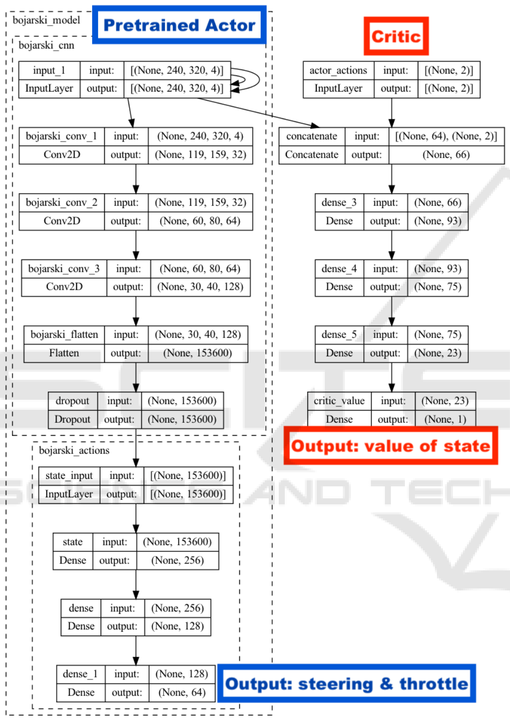

bojarski_model
Pretrained Actor
bojarski_cnn
input_1 InputLayer input: [(None, 240, 320, 4)] output: [(None, 240, 320, 4)]
bojarski_conv_1 Conv2D input: (None, 240, 320, 4) output: (None, 119, 159, 32)
bojarski_conv_2 Conv2D input: (None, 119, 159, 32) output: (None, 60, 80, 64)
bojarski_conv_3 Conv2D input: (None, 60, 80, 64) output: (None, 30, 40, 128)
bojarski_flatten Flatten input: (None, 30, 40, 128) output: (None, 153600)
dropout Dropout input: (None, 153600) output: (None, 153600)
bojarski_actions
state_input InputLayer input: [(None, 153600)] output: [(None, 153600)]
state Dense input: (None, 153600) output: (None, 256)
dense Dense input: (None, 256) output: (None, 128)
dense_1 Dense input: (None, 128) output: (None, 64)
Output: steering & throttle
Critic
actor_actions InputLayer input: [(None, 2)] output: [(None, 2)]
concatenate Concatenate input: [(None, 64), (None, 2)] output: (None, 66)
dense_3 Dense input: (None, 66) output: (None, 93)
dense_4 Dense input: (None, 93) output: (None, 75)
dense_5 Dense input: (None, 75) output: (None, 23)
critic_value Dense input: (None, 23) output: (None, 1)
Output: value of state

Figure 6: The resultant A2C model incorporating the pretrained behavioral clone model.

they incrementally trained to recognize hazard signs (demonstrating the ability to locate mock hazards). In addition, the team needed to figure out how the cloning model and the object recognition model could both share the same GPU at once. In addition, the team also determined how the rover would autonomously scan for these hazards when the A2C model was deployed. Eventually, YOLO3 was chosen as a simple bolt-on to our MAAP pipeline. We used transfer learning on a pre-trained YOLO3 model where the model was trained with the help of ImageNet and a collection of labeled hazard signs.

With the deadline for project deliverables looming the team was unable to produce a solution that would allow the A2C model and the YOLO3 model work on the same GPU, so YOLO was relegated to the CPU while the A2C was kept on the GPU. Overall, the model was able to function at a rate between 9 and 13 frames per second - a pretty good rate, considering there was no effort put towards optimized hardware performance.

The artifacts for the final two sprints were a final presentation to faculty members who were not involved with the project, a presentation at a local AI event, and a final demonstration of the hazard recognition mission to the project mentor. The mission was successful in that rover was able to locate 7 out of 9 easy-to-find hazard signs. The search routine was not optimal in that the program simply issued a command to the rover to perform a slow five-second in-place 360 turn to scan for hazard signs. This action was performed at one-minute intervals. Overall, the bolt-on object recognition portion of the project was rushed, but provided evidence that the students' approach to MAAP was plausible and had potential to scale for many real-world uses.

6 CONCLUSIONS AND FUTURE WORK

Through our MAAP capstone program, students were able to participate in developing an end-to-end solution involving advanced applied ML techniques at USAFA. Each student gained invaluable insight as well as improved ML skills, technical skills, and team-building skills employers would consider above average for newly minted undergraduate Computer Science majors. MAAP is a highly engaging program that allows students to experience how ML is designed, tested, improved, and applied in a real-world scenario outside of a simulated environment.

We feel that this program can be adapted to other robot modalities such as prosthetic limbs, aerial drones, and legged pack drones. Future plans include incorporating additional sensors and providing more interesting bolt-on missions and objectives for MAAP. We also plan to publish our data collections for use in research-related works as well as providing a publication covering the teaching materials for MAAP for use by other higher education institutions.

REFERENCES

Akbilgic, O. and Davis, R. L. (2019). The promise of machine learning: When will it be delivered?

Ardupilot (2022). The cube overview. https://ardupilot.org/copter/docs/common-thecube-overview.html.

Azari, A. R., Lockhart, J. W., Liemohn, M. W., and Jia, X. (2020). Incorporating physical knowledge into machine learning for planetary space physics. *Frontiers in Astronomy and Space Sciences*, 7:36.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Bojarski, M., Yeres, P., Choromanska, A., Choromanski, K., Firner, B., Jackel, L., and Muller, U. (2017). Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*.

Dominique Luna, Andrew Bauman, e. a. (2021). Udacity Behavioral Cloning Project.

Farjo, P. D. and Sengupta, P. P. (2021). Ecg for screening cardiac abnormalities: The premise and promise of machine learning.

Fraley, J. B. and Cannady, J. (2017). The promise of machine learning in cybersecurity. In *SoutheastCon 2017*, pages 1–6. IEEE.

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017a). Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2).

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017b). Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2).

Intel (2021). Introducing the intel® realsense™ depth camera d455. https://www.intelrealsense.com/depth-camera-d455/.

JetBrains (2022). The python ide for professional developers. https://www.jetbrains.com/pycharm/.

Kapoor, A. and Gardner-McCune, C. (2020). Exploring the participation of cs undergraduate students in industry internships. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 1103–1109.

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., and Pérez, P. (2021). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides, F. and Schmidt, B., editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press.

Lapan, M. (2020). *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more, 2nd Edition*. Packt Publishing.

Larsson, E. (2018). Evaluation of pretraining methods for deep reinforcement learning.

Lürig, M. D., Donoughe, S., Svensson, E. I., Porto, A., and Tsuboi, M. (2021). Computer vision, machine learning, and the promise of phenomics in ecology and evolutionary biology. *Frontiers in Ecology and Evolution*, 9:148.

Ly, A. O. and Akhloufi, M. (2020). Learning to drive by imitation: An overview of deep behavior cloning methods. *IEEE Transactions on Intelligent Vehicles*, 6(2):195–209.

McGunagle, D. and Zizka, L. (2020). Employability skills for 21st-century stem students: the employers' perspective. *Higher Education, Skills and Work-Based Learning*.

NVIDIA (2022). Nvidia jetson xavier nx for embedded & edge systems. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., and Peters, J. (2018). An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179.

Planch, B. (2019). *Hands-On Computer Vision with TensorFlow 2: Leverage deep learning to create powerful image processing apps with TensorFlow 2.0 and Keras*. Packt Publishing.

PowerBox Systems (2022). Radio system core. https://www.powerbox-systems.com/en/products/radio-system/fernsteuersystem/radio-system-core.html.

Python Software Foundation (2022). Python package index. https://pypi.org/.

Rutherford, S. (2020). The promise of machine learning for psychiatry. *Biological Psychiatry*, 88(11):e53–e55.

Sanner, M. F. et al. (1999). Python: a programming language for software integration and development. *J Mol Graph Model*, 17(1):57–61.

Schultz, L. (2014). Readability analysis of programming textbooks: Traditional textbook or trade book? In *Proceedings of the Information Systems Educators Conference ISSN*, volume 2167, page 1435.

Shapiro, R. B., Fiebrink, R., and Norvig, P. (2018). How machine learning impacts the undergraduate computing curriculum. *Communications of the ACM*, 61(11):27–29.

Smith, K. N. and Green, D. K. (2021). Employer internship recruiting on college campuses: 'the right pipeline for our funnel'. *Journal of Education and Work*, 0(0):1–18.

Smolkin, L. B., McTigue, E. M., and Yeh, Y.-f. Y. (2013). Searching for explanations in science trade books: What can we learn from coh-metrix? *International Journal of Science Education*, 35(8):1367–1384.

Sulmont, E., Patitsas, E., and Cooperstock, J. R. (2019). What is hard about teaching machine learning to non-majors? insights from classifying instructors' learning goals. *ACM Transactions on Computing Education (TOCE)*, 19(4):1–16.

Tai, L. and Liu, M. (2016). Mobile robots exploration through cnn-based reinforcement learning. *Robotics and biomimetics*, 3(1):1–8.

Toole, A. A., Pairolero, N. A., Forman, J. Q., and Giczy, A. V. (2019). The promise of machine learning for patent landscaping. *Santa Clara High Tech. LJ*, 36:433.

Torabi, F., Warnell, G., and Stone, P. (2019). Recent advances in imitation learning from observation. *arXiv preprint arXiv:1905.13566*.

Trister, A. D., Buist, D. S., and Lee, C. I. (2017). Will machine learning tip the balance in breast cancer screening? *JAMA oncology*, 3(11):1463–1464.

Valstar, S., Krause-Levy, S., Macedo, A., Griswold, W. G., and Porter, L. (2020). Faculty views on the goals of an undergraduate cs education and the academia-industry gap. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 577–583.

von Lilienfeld, O. A. and Burke, K. (2020). Retrospective on a decade of machine learning for chemical discovery. *Nature communications*, 11(1):1–4.

Winder, P. (2020). *Reinforcement Learning: Industrial Applications of Intelligent Agents*. O'Reilly Media.