Reactionary Targeting Defense System (RTDS)

Joseph Musante, Calvin Sands, and Chance Reimer

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The goal of this project is to provide a basis for future development related to tracking and targeting drones. By using computer vision, we are able to recognize a drone then fire a laser directly at it. Through extensible design, future developers will be able to use our project as a foundation to build upon by expanding its size and scope. The design uses a NVIDIA Jetson Nano to do image processing and sends targeting information to a servo PCB via UART. The custom-made servo PCB uses an atmega328p microcontroller to process UART data and control the servos to pan and tilt the camera and laser.**

***Index Terms* — Computer vision, convolutional neural networks, microcontrollers, artificial intelligence, embedded software.**

I. Introduction

In today's current age drones are becoming exponentially more popular for not just commercial purposes but also hobbyist and everyday folks as well. This rapid growth has raised some questions such as how we can go about protecting ourselves from the inevitable weaponization of drones.

With that said there is not much research or other similar noncommercial projects who are implementing a fully autonomous solution to this issue. Having a system that could be used in commercial or noncommercial environment is key which is why keeping the cost low was paramount. Ideally someone who is passionate about computer vision and image processing should be able to pick up our project where we left off by hitting the ground running with an entire targeting system already implemented. Our project will be able to detect a drone between one to five feet then by using geometry we will be able to get the location of the drone in the image. From there our servo system uses two servos one to pan and the other to tilt the camera and laser. Once the drone has been successfully targeted, we will fire a laser on the drone as a proof of concept for taking down the drone.

II. System Hardware

The system hardware chosen for the RTDS was selected for its balance of power and affordability. The goal was to develop the RTDS using practical, accessible components that would increase the reproducibility of the design, while also maximizing the capabilities of such a computationally expensive process that the RTDS would be performing.

A. Microcontroller

The microcontroller of choice for our project is the atmega328p for many different reasons. First and foremost, we wanted to choose this microcontroller because it is user friendly and also cheap. The atmega328p is unarguably one of the most widely used microcontrollers thus finding support for implementing our project will be as straightforward as possible. On top of that there are many libraries available for the Arduino Uno for controlling servos which is key when it comes to ensuring we are sending command signals as quickly and reliably as possible. UART on the atmega328p has 5V pins for Tx and Rx so we needed to use a logic level shifter to communicate to the Jetson Nano's 3.3V Tx and Rx pins.

B. Jetson Nano

The NVIDIA Jetson Nano is a very powerful yet affordable minicomputer with an embedded GPU which can be used for processing computer vision. The purpose of this computer is to process a constant feed of video and detect drones in it. From there it will perform calculations using some geometry to determine where the drone is in the image. From there it will send signals to the servo PCB via UART to tell the servos where to point to.

The current method to calculate the angle is based on a relationship between the drone's known size, and its width in pixels in the image. Using the premeasured drone length, a relationship can be formed with pixels and distance. A few assumptions must be made for the below equation to be valid, i.e. the drone must be moving orthogonally to the camera, and the drone must be perfectly enclosed by the bounding box generated by the YOLO model. Due to these assumptions not being met in practice, there is some error to the below calculation.

$$\Theta = \tan(\text{size\_distance}/\text{distance\_from\_drone})$$

C. Camera

The camera we chose to use to capture video and send to the Jetson Nano is a See3Cam CU30. This camera was a good choice for our project because it's supports a USB3.0 connection which enables its data rates to be up to 640 MBps which is more than enough for our requirements.

The camera is full 1080p, which is ideal for image processing for the Jetson Nano wouldn't be powerful enough to process full 4K video. The camera is powered from the Jetson Nano using the same USB3.0 cable that transfers data which is ideal being that it's only one wire. The final benefit to choosing this camera is given all its specifications it's also small and light enough to fit on our servo system mount. The size for the camera is as follows 40.7 x 40.7 x 33.4 mm. Weighing in at only 65 grams it's won't prove to be an issue for our servos to hold up over time too.

D. Servos

The servo of choice for our project is the SG90 9g by tower pro. Weighing in at only 9g it's the perfect choice because it barely adds any weight to our sensor system mount. Also, the servo provides enough torque to hold up our camera and laser without an issue. Another benefit of this servo is it's cost of only $5 per servo. Being that we need two of these servos one for panning and another for tilting it's key that they are affordable.

III. Device power

Being that we have no project requirements to make this design portable we decided to just stick with AC power. This means that we have 2 power plugs, one for the Jetson Nano and another for the PCB. Being that the Jetson Nano was a full-blown computer with an integrated dedicated GPU it was unreasonable to try to power it from a battery pack. With that in mind, it didn't make sense to try and use a battery for our servo PCB.

A. Jetson Nano Power

The Jetson Nano power supply is responsible for powering the camera as well. The power supply for the servo PCB is responsible for powering the servos as well as firing the laser too. There were a few different choices to provide power to the Jetson Nano. The first and easiest choice is though the 2-amp 5-volt micro-USB cable. The next straightforward approach to providing it power would be though the 4-amp 5-volt barrel plug jack. The last and most complicated approach to powering the Jetson Nano would be though the GPIO header pins. Although it's not recommended to only use a micro-USB cable to power the entire board and all its peripherals it's what we choose to do. This isn't recommended because depending on the type of peripherals used, they may draw too much power. However, given the peripherals we are currently using it hasn't given us an issue so far. If we were to use for example a more power-hungry camera or keyboard it may end up drawing too much power. [1]

B. Servo PCB Power

We considered using either a micro-USB cable or barrel jack plug to power the PCB. In retrospect, a micro-USB cable probably would have been a better choice being that they are easier to come by then the specific barrel plug we choose to use. We decided to use a 12-volt 6-amp barrel plug that is mounted directly to the PCB. One issue we faced was actually finding the barrel plug jack for the plug that was already mounted to the PCB. Turned out that the specific plug we chose to use isn't very common and could prove to be an issue if it was to ever break. With that in mind if we were to do a second prototype for the PCB, we would defiantly just use a simple micro-USB jack. With that in mind the 12-volts is stepped down using a voltage regulator to 6-volts for the servos. From there it's stepped down ever further to 5-volts for the microcontroller itself. [2]

IV. Servo Circuit Board Design

Our project required the ability to control two servos, one to pan the sensor system and another to tilt it. In order to control these servos, we needed a microcontroller that had easy access to reliable and user-friendly servo control libraries. On top of servo control we also needed the PCB to reliably send and receive data via UART. Last but not least the servo is also responsible for firing the laser at the drone once it's been successfully targeted.

A. Schematic Design

As seen in Fig. 1 below the main sub-systems of the PCB are the voltage regulators, logic level shifters and the microcontroller itself. There are a number of header pins as well used for powering and controlling the servos as well as the laser. We also have a couple other header pins dedicated for programming and testing the microcontroller.

B. PCB Design

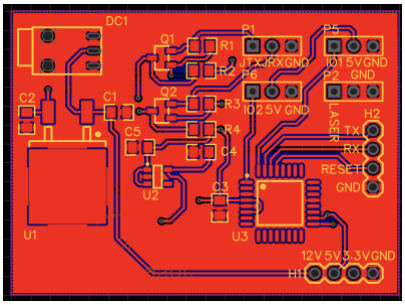


Figure 1: Schematic of PCB Design

As for the actual design of the PCB being that we didn't have any size requirements it wasn't too much of a

challenge. The main concern being that we were using UART as the only data transfer we needed prioritize ensuring it was as reliable as possible. This meant minimizing noise, the way we went about achieving this was ensuring that the traces were as shot as possible between the logic level shifters and the microcontroller. This effort can be seen upon looking at the component layout in Fig. 2 below. We decided to use all surface mount components to make it look nice and to just keep the build consistent.

C. Logic Level Converter

For our project a strong UART connection was paramount in order to be able to send and receive data from our Jetson Nano. This meant that we needed to implement the logic level shifters on the PCB instead of using breakout circuit board in order to minimize extra wiring which could cause more loss and noise.

The logic level converter is used between the custom servo PCB and the Jetson Nano. Its purpose is to step up the voltage from the Jetson Nano the UART signals from 3.3V to 5V for the atmega328p on the PCB. It also is used to step 5V UART signals from the PCB back down to 3.3V on the Jetson Nano. We used a breakout version of the schematic below first to test the design then implemented two of the same circuits on our PCB. The only real issue we found with the logic level converters is that over time they may not be as reliable as would have liked them to be. With that said, if we were to make another version of the PCB we would use more expensive components. The circuit diagram for a single logic level converter can be seen in Fig. 2 below. [3]

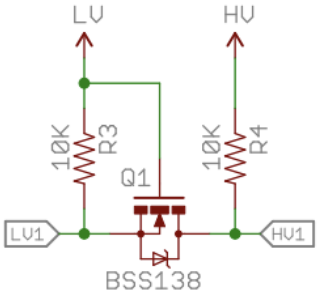


Figure 2: Logic Level Converter

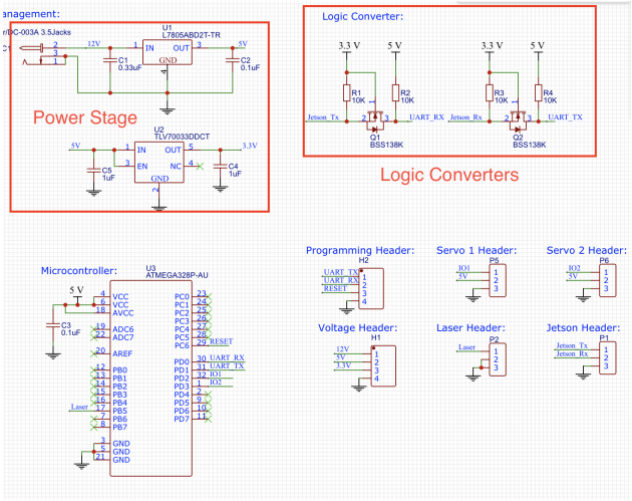
D. Voltage regulators

As for powering the various components on our PCB we decided to use two different voltage regulators. This was necessary because our microcontroller needed an input voltage of 5 Volts while our servos operated most reliability at 6 Volts. In order separate the input power out like this we decided to have one very large voltage regulator which can be seen in figure 2 on the bottom left side of the PCB. This first voltage regulator was responsible for stepping the input voltage from 12 Volts down to 6 Volts for the servos, we chose to use a Stmicroelectronics chip called the L7805ABD2T-TR. [4] This chip was an important choice because needs to provide enough current power to not only the atmega328p but also our two servos. [5] Being that the chip provided up to 1.5A it supplied enough for everything we needed. The second voltage regulator we chose was the Texas Instruments chip called the TLV70033DDCT. It was responsible for taking the 6 Volts from the first voltage regulator and stepping it down to 5 Volts for the atmega328p. The following power stage with voltage regulars can be seen below in Fig. 3.

Figure 3: PCB Circuit Diagram

V. Software

The RTDS uses a software pipeline that aims to



efficiently convert data captured by the sensor camera and convert that into movement data given to the servos. Through the use of a sophisticated and "lightweight" object detection algorithm, alongside a series of embedded data parsing algorithms, the RTDS will be able to recognize its nearby targets and successful output tracking information.

A. YOLO (You Only Look Once)

The primary purpose of the RTDS is to be able to identify and track flying nearby quadcopter drones. To the extent of accomplishing this goal, an object detection

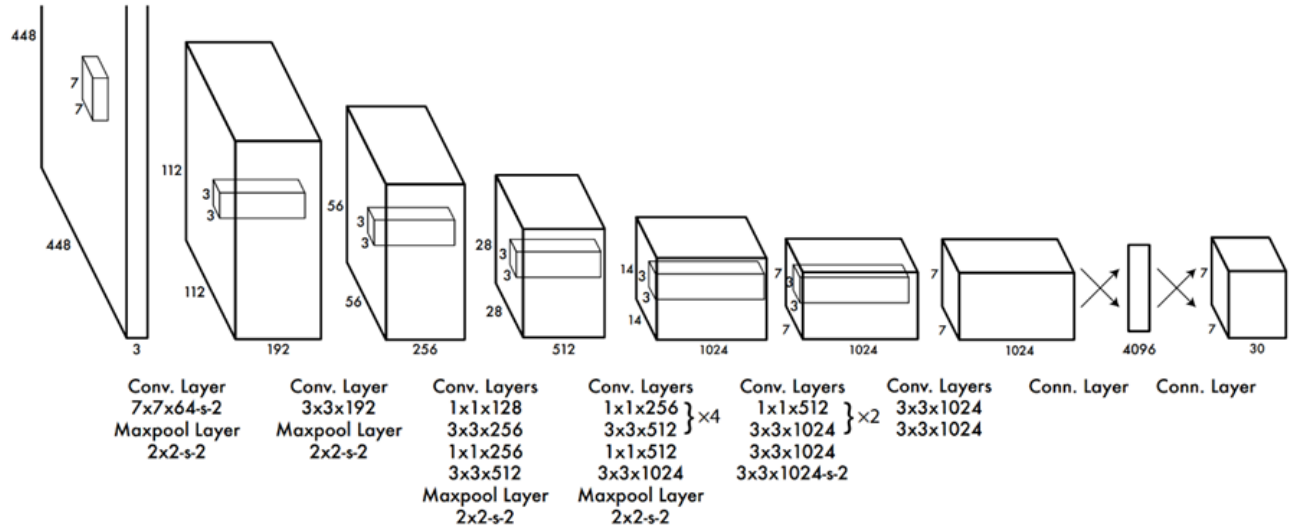


Figure 3: The YOLO architecture. Courtesy of Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi.

algorithm is necessary in order to provide the device with the ability to distinguish the targeted drones in their viewfinder. A small variety of algorithms that were considered lead to YOLOv3 (the third iteration in its design) being the algorithm chosen for this role due to its advantages.

One of these advantages includes its extensive documentation. YOLO has become a very popular object detection algorithm, and along with this popularity has come a wide degree of documentation for its use in diverse systems and environments. The existing resources provided the team the ability to develop and use this algorithm on a variety of different operating systems, and to be able to develop its utility to exactly match our specifications and requirements.

Another advantage was out-of-the-box usability. By virtue of being able to start using the tools that implemented the YOLO algorithm early in the design and development process, it was anticipated that the RTDS team would be able to identify problems early in the training and testing processes and would have ample time to both resolve those problems and optimize their solutions.

YOLO at its core is a CNN (convolutional neural network), as demonstrated in Fig. 4, that uses a novel means of observing the entire given image both during training and during trials. Most CNNs perform what can be described as "scanning" the target image. At each and every pixel interval they search for shapes and objects that match that of a previously learned model. This approach can be very time-consuming, however, as one can imagine the amount of time (relatively speaking) it takes to perform a full network of convolutions multiple times on multiple regions, versus one single time against the entire image. [6]

Not only does this newer approach improve the speed of the network, but it also improves the accuracy. By looking at the entirety of an image during training, a model learns not only what the target object looks like, but also what it does *not* look like. The YOLO model simultaneously learns both the positive identification of the target object, as well as the negative space around it, giving it a more robust capability to parse a target object from a cluttered environment or background.

With the RTDS being a smaller constrained system, however, even the relative efficiency of YOLO originally proved to be too taxing of a process to run at a functional rate. YOLOv3 uses a 106-layer full CNN, and initial tests using it on the RTDS's Jetson Nano saw the process rate sit at around a useless 1-2 FPS (frames-per-second). Tiny YOLOv3 is an adapted version of the YOLOv3

TABLE I
YOLOV3 PERFORMANCE COMPARISONS

Model	Train	Test	mAP	FLOPS	FPS
YOLOv3-320	COCOtrainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCOtrainval	test-dev	55.3	65.86 Bn	35
YOLOv3-608	COCOtrainval	test-dev	57.9	140.69 Bn	20
YOLOv3-tiny	COCOtrainval	test-dev	33.1	5.56 Bn	220

architecture, designed for solving this problem. As a mere 24-layer CNN, Tiny YOLOv3 suffers from a reduced accuracy, but benefits from a much-increased processing rate. These differences are observed in Table 1 [7], comparing the mean average precision (mAP) to the FPS of various implementations of YOLOv3, including different CNN network resolutions of the standard 106-layer YOLOv3 (i.e., YOLOv3-320 having a network resolution of 320x320). As witnessed, the speed increase of YOLOv3 greatly outweighs the lost accuracy, with Tiny YOLO (YOLOv3-tiny) performing at about 64% the accuracy of YOLOv3-320, but doing so nearly 5-times more quickly.

B. Training YOLO

The approach that the RTDS required in order to train its YOLO model was an atypical situation. It was evident from the beginning of the design steps that the RTDS would require a new, uniquely trained model in order to detect the quadcopter drones it would be targeting. In order to train such a model, however, a dataset needed to be acquired. This would be accomplished either by finding an existing open-source dataset, or by following the painstakingly long path of curating a new one from scratch.

YOLO datasets have two requirements in general: An image containing the object that is to be identified, and a parallel text file containing the name of the class of that object and its bounding-box's coordinates with the image. One such dataset was discovered early, called Drone-net, maintained by Chuan-en Lin on Github [8]. Initial models trained from this dataset however discovered a very evident misalignment in the goals of the RTDS versus the style of images used by Drone-net: The images of Drone-net were all low-resolution, centered, clear images of a variety of quadcopter drones. They were all prominently displayed in their photos, most often being the obvious focus subject. YOLO models trained from this dataset were quick to identify similarly-viewed drones, but had a lot of difficulty identifying drones from any reasonable distance away from the camera (such as more than two or three feet), or drones that were not in good lighting, in focus, or partially obscured by foreground objects. This made it evident to the team that a new dataset would need to be built using images of drones that more closely matched the types of environments the RTDS would be used in – at a distance, with moving subjects, that might not be in the center of the image.

C. Building the YOLO Dataset

Creating such a dataset required simply gathering imagery from realistic environments and scenarios from

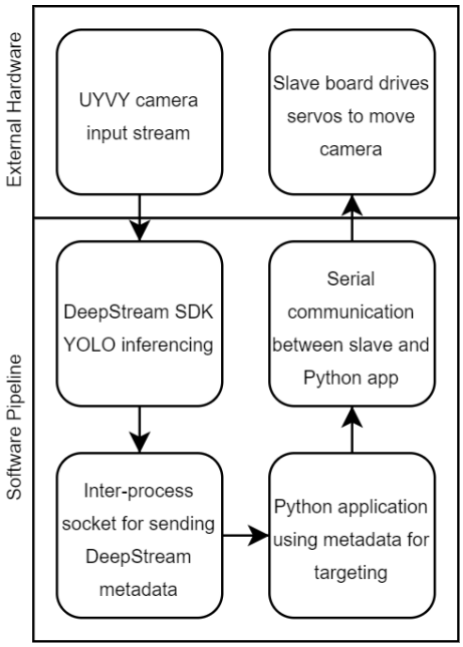


Figure 4: Tracking Algorithm Pipeline

the perspective of the RTDS of a nearby flying drone. These images were taken from various angles and distances, using various lighting levels and backdrops. Different cameras and resolutions were also used to capture the widest possible variety of images of the target drone. These images were then each annotated with a bounding box to denote where in the image an object was, and what the class (or name) of the object was. As of the writing of this document, the most recently trained YOLO model used a dataset containing a little over 2300 different images, which is just cresting the minimum recommended dataset size of about 2000 unique images [9]. This dataset continues to grow however, as the team discovers new environments in which the RTDS underperforms at identification. Thus, new images are taken from the troublesome environment and provided to a newly trained model.

Models trained using such a paradigm have since been a huge success, with each new model increasing the robustness of the RTDS's tracking capability.

D. Tracking Algorithm Pipeline

Using YOLO to track the drone only proves to be half the battle for the RTDS. Once the YOLO model had been sufficiently trained and implemented on the Jetson Nano, a new pipeline needed to be developed in order to translate those outputs from YOLO into an understandable format such that the Arduino control board can provide directions to the servo mounts. Fig. 5 provides an overview of all the components of this pipeline in a narrative order.

The pipeline begins with using a third-party development kit to optimize the actual YOLO runtime on

our development board. YOLOv3 is a computationally intensive algorithm and runs in-optimally slow on the Jetson Nano platform, at around 1-2 FPS. Even with the less intensive Tiny YOLOv3 implementation, the Jetson Nano FPS rate was insufficient to track objects in real time. The solution was to use NVIDIA's DeepStream SDK which is optimized for the Jetson Nano's GPU. This, combined with using Tiny-YOLOv3, provided the means to be able to consistently achieve about 24 FPS during runtime.

The DeepStream SDK is built using the Gstreamer platform, meaning that processing occurs through a pipeline of plugins that perform actions on data. It comes ready with prebuilt plugins for tracking and applying machine learning algorithms, which simplifies development. Developers using these tools are able to create additional plugins that are more customized to their system's needs. These tools provided the groundwork on which YOLO was to be implemented on the Jetson Nano.

The inter-process communication is handled using ZeroMQ. ZeroMQ is a universal messaging library, utilized to send messages over local addresses, or to other computers or servers [10]. To simplify the purpose of each process, it was decided to have the Deepstream pipeline send useful metadata over a local port to a listening python application responsible for commanding the slave board. To further increase speed, Google protocol buffers were utilized to serialize the metadata, decreasing packet size.

The Python communication application uses the metadata sent by the DeepStream pipeline to generate the relevant data to be read and understood by the slave microcontroller board. These instructions include the tracking and movement instructions for the servos, as well as when to fire the laser. It also bears the responsibility of determining when a tracked object has left the maximum field of view. Additionally, it notifies the user of when tracking is in progress, or when a hit occurs.

D. Slave IO Board

The utilized metadata includes the size of the captured bounding box, the location of the bounding box within the current image, and the type of detection (the label of object detected – for the purposes of the RTDS this will always be a drone). The application utilizes the most recent two detections to approximate the speed and direction of the tracked target. This is possible due to having the approximate size of the drone being tracked as a known value. By using the known size of the drone, as well as the relative size of the bounding box, the distance of the drone (how far it is from the RTDS) can also be algebraically approximated. All these values, either known or measured in real-time, are used for commanding the Slave IO.

The Slave IO uses a communication protocol specifically defined by the RTDS team for transmitting the relevant targeting information from the metadata. It uses a word length of 4 bytes, consisting of two 16-bit signed integers, describing the step size and angle relative to the x-axis. Both values are used by the Slave microcontroller to translate into a direction and a speed to move the camera. In return, the Slave IO board is responsible for notifying the Python application when it approaches the edge of the viewing window. For firing the laser, a reserved word is used, which is the only communication for which the system requires an acknowledgement.

Finally, the Slave microcontroller carries the responsibility for sending the final movement instructions to the servos. It also is responsible for monitoring the viewing angles and window of the servos and provides notification if the commands it is receiving are approaching the extreme edge of its field of view. The libraries for serial of the ATMEGA328p chip on the Arduino Slave microcontroller, however, do not allow for access to any underlying API, meaning that interrupt-based serial interfaces were difficult to implement. A solution to this was found in protothreading in order to grab the information from a communication socket, and for sending and receiving serial information.

VI. CONCLUSION

We firmly believe that our project will pave the way for the future when it comes to the personal and commercial industry protection against drones. Whether it's a rogue drone or one being used to spy on someone our project is a fundamental building block for future endeavors in this field. Being able to not only recognize drones in an image or video but also track them is a monumental achievement for 3 undergraduate students to have accomplished.

For starts, the choice of using the powerful yet affordable Jetson Nano to do image processing was brilliant. For it allowed us to do everything we needed when it came to not only image processing but also rapid prototyping and development. The choice of using YOLO allowed us to save time by not reinventing the wheel of neural networks in computer vision but allowed us to train our own model to confidently detect drones. The choice of our simple yet powerful atmega 328p microcontroller allowed for easy programming and implementation. All in all, our group worked together just as well as our hardware did which enabled us to have a successful project in the end.

ACKNOWLEDGEMENT

The authors would like to thank Dr. Lei Wei with his continued assistance and support throughout this project.

THE TEAM:

Joseph Musante will be graduating with a Bachelor of Science in Computer Engineering. He spent a year and a half working as A CWEP for Lockheed Martin in Systems and Software Engineering. Also interned with Northrop Grumman and accepted a full-time job there for software engineering once he graduates.

Chance Reimer will be graduating with a Bachelor of Science in Computer and Electrical Engineering. He has had internships at Eizo Rugged Solutions, Northrop Grumman, and was a Coop at Siemens Power and Gas. Chance's main interests include embedded engineering, computer vision, and FPGA design.

Calvin Sands will be graduating with a Bachelor of Science in Computer Engineering. He has spent the previous two years as an intern with Lockheed Martin as a Systems Engineer and a Software Engineer. He has recently accepted an offer to continue full-time in his current position as a Software Engineer upon graduating.

REFERENCES

[1] Mitch Allen, "JETSON NANO POWER SUPPLY (BARREL VS. MICRO USB)" desertbot.io, [Online]. Available: https://desertbot.io/blog/jetson-nano-power-supply-barrel-vs-micro-usb. [Accessed 2 April 2020].

[2] "ATmega328", Microchip, [Online]. Available: https://www.microchip.com/wwwproducts/en/ATmega328. [Accessed 8 December 2019].

[3] "N-Channel Logic Level Enhancement Mode Field Effect Transistor" Fairchild Semiconductor, Oct 2005. [Online]. Available: https://cdn.sparkfun.com/datasheets/BreakoutBoards/BSS138.pdf. [Accessed 13 February 2020].

[4] "Positive voltage regulator ICs" stmicroelectronics, [Online]. Available: https://datasheet.lcsc.com/szlcsc/1810010314_STMicroelectronics-L7805ABD2T-TR_C86206.pdf. [Accessed 9 November 2019].

[5] MertArduino, "How to Use a Servo Motor With an External Power" instructables, [Online]. Available: https://www.instructables.com/circuits/. [Accessed 26 November 2019].

[6] Redmon, Joseph, et al. "You Only Look Once: Unified, Real-Time Object Detection." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, doi:10.1109/cvpr.2016.91.

[7] Redmon, Joseph. Darknet: Open Source Neural Networks in C. Available: https://pjreddie.com/darknet/.

[8] Chuan-en Lin. "Chuanenlin/Drone-Net." GitHub, 25 May 2019. Available: https://github.com/chuanenlin/drone-net.

[9] Alexey. "AlexeyAB/Darknet." GitHub, 26 Oct. 2019. Available: https://github.com/AlexeyAB/darknet.

[10] Hintjens, Pieter. "ØMQ - The Guide." ØMQ - The Guide, zguide.zeromq.org/page:all.