Quadcopter Object Recognition: an ME 740 Final Report

Richard Heath

5/6/2022

Abstract

An object recognition and tracking system was configured with ROS Melodic on a Jetson Nano Developer Kit, using a RealSense D435i depth camera to experiment with object classification, and some aspects of general autonomous environmental awareness. This is intended to be used in autonomous navigation and delivery in a quadcopter application.

1 Introduction

Real-time autonomous mapping and classification tasks are becoming more relevant and common in modern robotic platforms. Simultaneous Localization and Mapping (SLAM) is one advanced method that autonomous robots can use to save, store and recall obstacles in a map, real-time as they navigate through it (1). This can become crucial in fast and efficient motion planning algorithms that take into account boundaries, cost functions, and more. It also enables autonomous vehicles to be aware of their surroundings in GPS-denied situations (2). The innovation of compact devices such as the NVIDIA Jetson family bring the capabilities of an AI computer into a module about the size of a cell phone, weighing around one pound. This gives mobile autonomous robots opportunities to recognize their surroundings and make independent decisions based on them. This can be identifying an obstacle, a target, or mapping an unknown environment, real-time.

One of the biggest benefits of both SLAM and image classification is that they can work in both static and dynamic environments, since they can update the map or the object real-time whether the objects in mind are stationary or in motion. I wanted to explore the idea of mapping and object recognition that could be applied to autonomous vehicles that wanted to perform a mission where they wanted to plan and maintain motion such that they are above a specified target, such as a hostile vehicle if carrying a lethal payload, or simply a landing pad for autonomous landing. The landing could be dynamic, such as landing on a moving vessel on land or at sea. If successful, this could become an open source tool and tutorial that could advance research and development in fields such as defense, mapping, and more.

Since most of the technological advances are recent, there are not many public projects that are similar. One of the most similar projects is fast object avoidance with quadcopters equipped with a Jetson Nano and a Logitech camera (3). Another project proposes quadcopters equipped with a Jetson Nano and a Raspberry Pi to perform tasks such as tracking and recognizing people in a particular area, such as a quarantine zone (4).

2 Hardware

For the materials and inspiration needed to explore this, I reached out to my former quadcopter team advisor May-Win Thein at the University of New Hampshire. Her graduate and undergraduate hybrid QuadX Swarm team had hardware available for use. This included an NVIDIA Jetson Nano Developer Kit, a RealSense D435i depth camera, and a Raspberry Pi 4. The Raspberry Pi is used in their quadcopter system, but was not used in this research. Their quadcopters currently use ROS on the Pi to utilize a Pixhawk GPS system, and use Xbee modules for cross-quad communication. Their research focuses on optimizing swarm algorithms to conduct multi-quadcopter missions.

The Jetson module is an AI-capable module that my employer in the defense industry currently considers in their own research and development efforts. The Nano is the most affordable and least powerful of the Jetson family. It weighs 0.550 lbs. It has 4GB of RAM, and operates on Linux Ubuntu 18.04 Operating System. I flashed the Nano image to a 64GB microSD card, and installed it into the Nano. The SD card holds the Nano configuration. In theory, different SD cards with different configurations can be used with the same Jetson device, which makes switching, backing up, and testing configurations easier with multiple SD cards. I only used the one, but I did take note of this. Jetsons are capable of having WiFi and Bluetooth implemented easily and affordably, but for this early development research, only ethernet connection was utilized. The Nano is capable of Headless mode, where a monitor, keyboard, mouse, and display are unnecessary if the user SSH (Secure SHell) connects to the Nano, similar to a remote desktop application. Since I did not want to install Linux on my PC, I used PuTTy as a terminal interface and Xming as an X11 forwarding, for porting to my computer's video display. One benefit of this is the convenience of working on the Nano anywhere in the proximity of the WiFi network connection. Another benefit is less wires (keyboard, mouse, display), which saves on space, tidiness, and most importantly less power needed by the Jetson. Following advice from tutorials and a lesson learned from errors of the RealSense camera, I unplugged the microUSB 2A cable that powered the Jetson, and bought a 4A barrel jack. More power, and less connected devices to power.

The RealSense D435i depth camera has several operating modes, which can transmit data simultaneously. There is an RGB (color image) mode, a stereo (depth) mode, and two IR modes (infrared 1 and infrared 2). The depth mode can be used to record a point cloud, which can be utilized in SLAM functionality. The infrared has the advantage of detecting objects in the array of lasers in environments that the depth and RGB cameras would not be accurate, such as dark environments. This would likely require additional training for models to process infrared images. The "i" in D435i indicates that the camera has a built-in IMU (Inertial Measurement Unit) system in it. Three axes of accelerometer and gyroscope data can be pulled directly from the camera. The camera was mounted to a tripod for this initial work, but could be removed from it and fastened to a custom 3D-printed mount that utilizes its one internally threaded mounting hole. It weighs 0.661 lbs, and could be replaced by lighter (likely more expensive) cameras.

ROS (Robot Operating System) Melodic was installed on the Jetson Nano. This seemed to be the most supported ROS distribution for the Nano, especially on Ubuntu 18.04. An open source ROS Wrapper from RealSense was installed in order for ROS, i.e. RViz video application, to subscribe to topics available in the RealSense camera. The cameras and IMU can be accessed and published from there. My first attempt to install the ROS Wrapper failed. I had followed a "JetsonNanoHacks" YouTube and Github tutorial (5) to install this and previous packages. Basically downloading and running scripts for several installations. I learned that since the tutorial, multiple new releases of several dependency packages had occurred, so the scripts may have included obsolete packages. I

formatted and flashed the Nano SD Card and installed all of the packages directly from the source, which helped. This troubleshooting set me back in the beginning, but ultimately helped deepen my understanding of the software.

Figure 1: Hardware used in this study. The Jetson Nano (left) is a computer with AI capabilities. The RealSense D435i camera (right) gives color, infrared, and depth images.

3 SLAM

With ROS Melodic installed, I confirmed that the Jetson could subscribe to the RealSense camera topics (RGB, depth, infrared 1, infrared 2) using RVIZ. RVIZ is used in a lot of the open source applications to visualize data, including images, point clouds, maps, and more. The user can view the raw 2D image input, while in the RVIZ application exploring the 3D point cloud of that image in 3D space. The SLAM application is supposed to utilize the IMU of the camera to save features on the RVIZ map. As the camera moves around slowly, the map can be saved. The saved map can be written to an image file on the Nano. This may not be ready for quadcopter use, as the tutorial states that fast movements may make the algorithm lose its place and restart the map.

I was able to get the live image portrayed in RVIZ [Figure 2]. Unfortunately, I was not able to get it to save and add to the map as I moved around. I think there could be more work to do in defining the map in the RVIZ workspace, which I am still new to. It was still good to confirm that the point cloud and other settings worked. I was able to save a rosbag file locally, which was another useful skill to have for storing and recalling data.

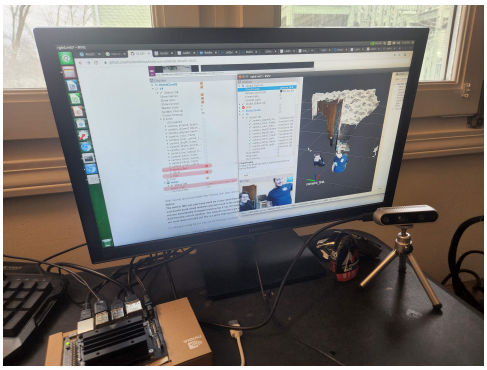

Figure 2: RGB point cloud in RVIZ during a SLAM demo attempt.

4 Object Classification

The main appeal of the NVIDIA Jetson family is their ability to perform high-end computations for machine learning in a relatively small package. Preconfigured packages exist for image processing in Linux using Python, taking advantage of databases and models that are publicly available. Imagenet and Googlenet are two of the existing databases and models that can be implemented on the Jetson Nano. These typically look at image files, or video files frame-by-frame, to detect labels that the models have been trained to identify. In Figure 3 an example image file of a jelly fish is classified. This can be done frame-by-frame with a video for the jellyfish as well.

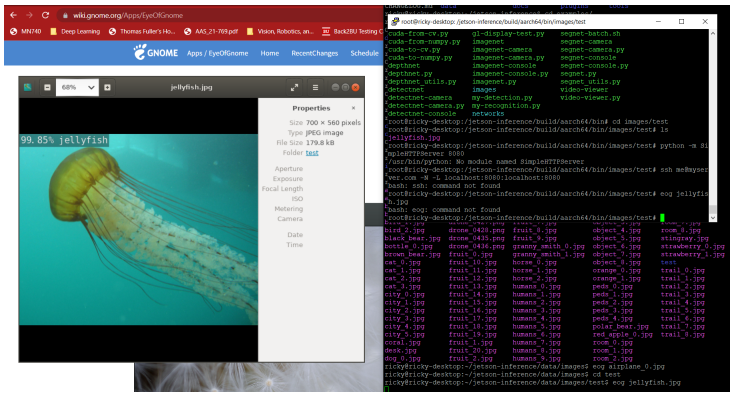

Figure 3: Output label laid over a pre-existing example image of a jellyfish. This new image is saved as part of the script.

Next, I followed a tutorial that processed images on an architecture known as DetectNet [Figure 4]. DetectNet was trained with minibatches of images and labels, using a transformation layer to apply online data augmentation. Feature extraction is performed by a GoogleNet Fully Convolutional Network (FCN), and object and bounding boxes are predicted (6). Simultaneous error measurement via L2 Loss on the predicted coverage map and L1 loss on the predicted bounding boxes (7). These losses are summed before updating the weights of the model and performing backpropogation. The model is validated with validation images, where mAP (mean Average Precision) is the evaluation metric chosen to compute and compare performance. Precision is defined by the equation

$$\frac{TP}{TP + FP}$$

where TP are the True Positives (model prediction equals ground truth label) and FP are False Positives (model prediction does not equal ground truth label).

Since PyTorch was installed on the Jetson Nano, I had the tools necessary to create a Python script or notebook that would take this pretrained model and add image labels to it based on additional training and validation images and labels that I could compile and feed into it. This is important for customizing the autonomous vehicle object recognition for custom targets, such as hostile vehicles for targeting or precision landing on a prescribed landing pad shape. I had the general knowledge and skills from Machine Learning and Deep Learning courses I attended at Boston University. However, this would require more time and effort than was available for the course of this project, so I noted the

capability and focused my efforts on proving the concept of tracking the detected objects in the 3D space (XYZ coordinates) that a quadcopter would operate in.

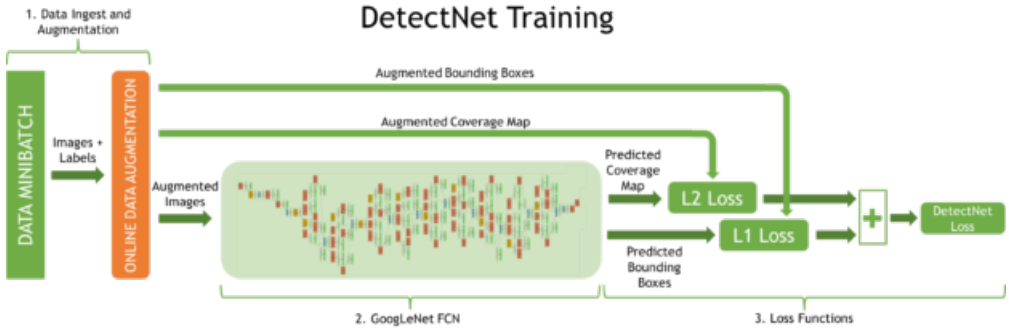

Figure 4: DetectNet training flowchart.

After getting this demo to work, I took it to the next level with ROS Melodic integration. This demo utilized the ROS Melodic interface. I was not able to get this to work with SSH and X11 forwarding, so I had to view it directly from the Jetson to confirm it was working [Figure 5]. However, without the video display, the terminal still printed information rapidly, including the target label (i.e. "person") and confidence level. This information can still be used remotely and pulled into ROS or other interfaces for making decisions and planning the quadcopter motion path and mission plan.

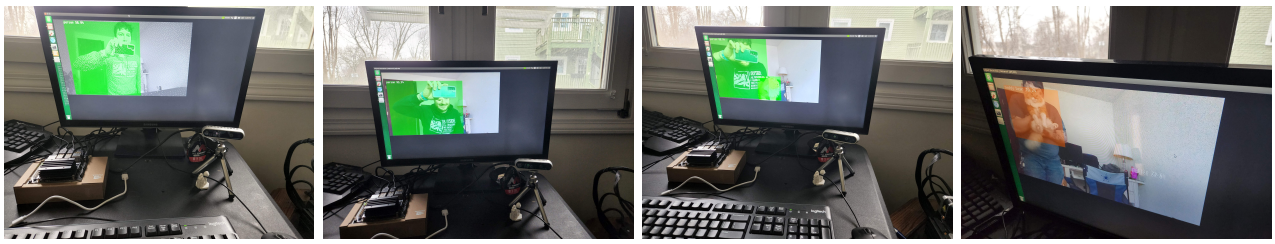

Figure 5: Jetson inference demonstrations I performed. One of 90 labels is chosen for objects the model is pretrained to classify. The confidence percentage and bounding box are overlayed on the input image, and data is published to ROS nodes.

5 DetectNet to Quadcopter

Once image classification was implemented to the Nano in ROS Melodic, I knew that would become the key to object detection and tracking from a vehicle. One of the goals that the UNH quadcopter team had not attempted was identifying and tracking a landing pad. This could enable the quadcopter to land on another vessel in a network of autonomous vehicles, such as a boat or rover. For a proof of concept approach, I took the DetectNet label 13, a stop sign, and printed it out [Figure 6]. This would emulate a landing pad. Traditionally, landing pads have high contrast in both color and geometry compared to their surroundings, such as a yellow H and circle. Hence a stop sign makes sense, and could easily be installed in an application such as landing a quadcopter on a sea vessel, which the

quadcopter team would eventually like the capability for.

The DetectNet ROS implementation creates ROS topics as outputs, and prints information in the ROSOut ROS topic, which is echoed in the main terminal. I wanted to listen to that broadcast, calculate what the displacement between the object and the camera, and then publish it to a new ROS node for the quadcopter to subscribe to. The quadcopters utilize Pixhawk GPS hardware, which takes XYZ waypoints and tells the quadcopter to autonomously navigate to them, without awareness of its surroundings.

By following the ROS tutorials on the website, I was able to create my own messenger Python script on the Jetson [Figure 7]. This script subscribes to the DetectNet output to listen for any detected objects. It then parses the data for the image label ID, the confidence percentage score, and the XY coordinates (in pixels) of the centerpoint of the detected object. In the Python script, if the label ID is 13 (a stop sign), then the parsed data is published to a new ROSTopic that I called "targetData". I added a conditional that will only publish if the model is above a certain confidence threshold, i.e. 75

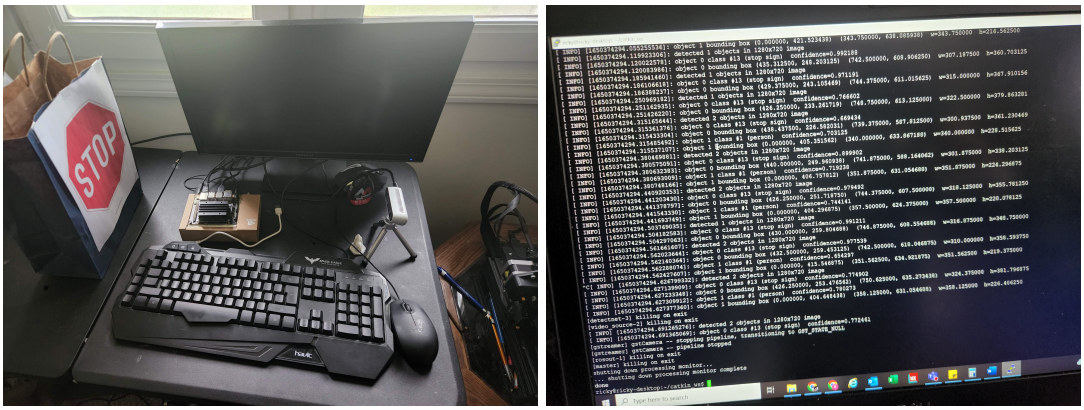

Figure 6: The stop sign test setup (left), and the default ROS topic ROSOut output echoed in the terminal (right).

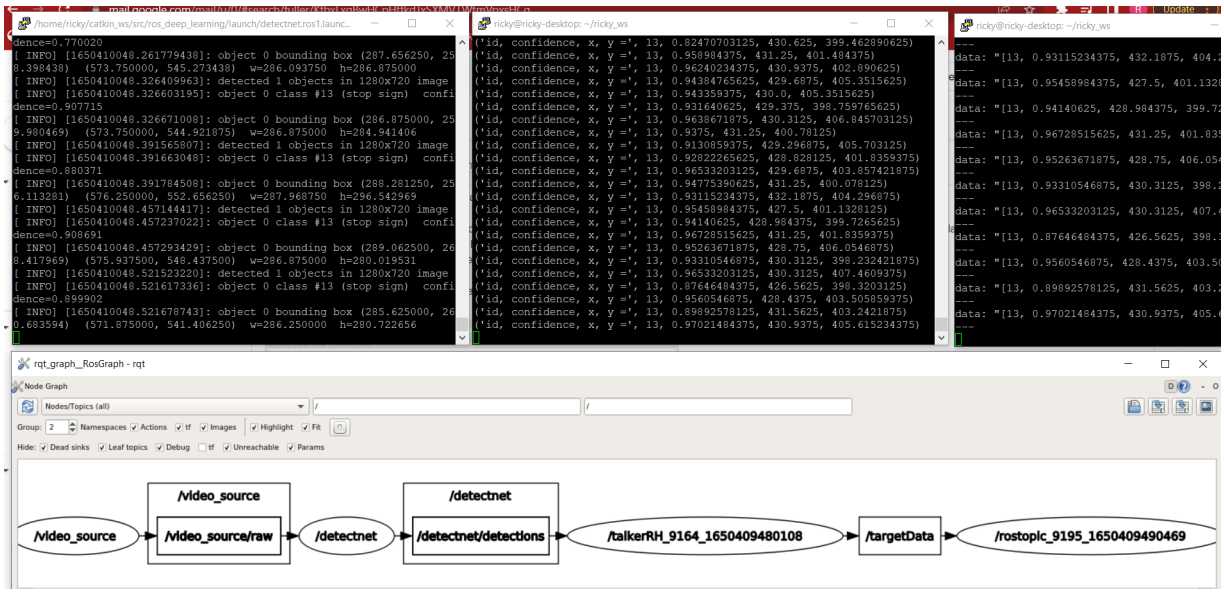

Figure 7: Raw DetectNet ROSOut echo (left), Messenger script data parsing processing and publishing (middle), and a terminal that is set to ROSTopic Echo to prove that the messenger script is publishing. QRT Graph (basically a ROS Topic Flowchart) from raw video source to the ROS Topic published by the Python messenger script (bottom).

6 Pulling Everything Together

After proving the DetectNet data was able to be listened to, processed, and published to a new ROS node that the quadcopter could tune into, I wanted to make this process more streamlined. The end goal is to implement this into an autonomous quadcopter, so the less user input, the better. I realized that by creating a bash script, I could run one script in the Linux terminal that would open two additional terminals and run ROS commands. This meant that with a couple keystrokes I could launch my bash script, and the bash script would take care of the ROS Launch file for DetectNet, as well as launching my messenger Python script. I even added the "rostopic echo" command to the bash script's terminal so the user could verify that the messenger Python script was publishing the processed data.

Next, to take the detected object center point XY coordinates and find out how far away that object is, I had more work to do with both ROS and Python. I came to the unfortunate realization that the DetectNet demo and the RealSense D435i camera ROS demo could not occur simultaneously. Even though I could launch both the ROS SLAM and DetectNet launch files at the same time, which published the ROS nodes of both demos, the to demos called for data from the camera in different ways such that they could not coexist. So even though the depth camera published ROS topics for the data, there actually was not data in there while the DetectNet demo was publishing object recognition data. The DetectNet demo read the image stream directly from the V4L2 (Video 4 Linux 2) path dev/video2, which was the RGB camera. When reading from this, the RealSense camera launch could not talk to the camera, and vice versa. I realized that ultimately to take the DetectNet and process it to meaningful XYZ data, the best path forward would be to save image files from the RealSense ROS demo locally, then feed them into a DetectNet script that takes image files instead of direct video

feed. The DetectNet demo documentation describes being able to feed it video or image files instead of live video feeds, so I assume this would still publish a ROS topic about the detections the same way the live demo did. From there I could simultaneously parse the depth camera raw data. I was able to implement a function online that converts that data to a distance in millimeters. The first remaining obstacle is to get the depth and RGB data simultaneously for DetectNet processing and depth calculations. Based on the proposed orientation and mounting of the camera, this would give x pixel and y pixel coordinates of the centerpoint of the detected object, and a z coordinate of a depth in millimeters. If

If I have time to implement this, I would like to make a script that subscribes to the RealSense ROS demo to save RGB camera raw data. This should be saved to a file and fed into the DetectNet algorithm to see if any objects are detected, similar to the demo. Ideally the feed could read live from the RGB raw camera output ROS Topic. But since the DetectNet seems to want either a live direct V4L2 feed or an image file, I would have to settle for the image file From there, whether one or many objects are detected, if one of the objects is the desired one (label 13, a stop sign, in this case), then the script could subscribe to the RealSense ROS

7 Concluding remarks

There are many open-source tools available for the Jetson Nano. I was able to install and test examples of SLAM and image classification, both of which are within ROS Melodic. SLAM could use more troubleshooting, but the dependencies functioned properly. Image classification could pass information through ROS to other applications, which is crucial for navigation, flight controller, and path planning.

I was also able to install ROS2 Dashing, which is crucial due to ROS1 nearing its end of life in the next few years. More recent ROS2 distributions did not have a lot of forum support, even with Dashing, so I stuck with ROS Melodic as a baseline. If given more time, I would like to convert the SLAM and image classification to ROS2. These could be later implemented into platforms such as the quadcopters, to help them recognize their surroundings.

Installing Python tools such as Pytorch gave me the ability to create, train and test my own machine learning models. This could be used for custom image classification models, such as recognizing landing zones, targets, and obstacles. One PyTorch-supported concept, which I am exploring currently in my Deep Learning group project, is Optical Flow. Optical Flow which analyzes images to track pixel displacement, and therefore object motion with respect to the viewer. My group is trying to track subpixel movement with Recurrent All-pairs Field Transform (RAFT), a promising new CNN and RNN model that has been introduced around 2020 (8). I believe that could be another route to environmental awareness and object recognition and tracking, and would have liked to explore that concept if I had more time.

By creating my own Python and Bash scripts, I was able to streamline the DetectNet ROS demonstration. This would be useful in the application because the user can run it with one command in one terminal, and the application is meant for autonomous unmanned vehicles.

I was grateful for this opportunity to learn about and interact with the Jetson device, Linux, and ROS, none of which I had prior experience with. I am also grateful to have had the help of my colleagues from the University of New Hampshire, who lent me the hardware and helped me troubleshoot and learn efficient Linux use. My employer has programs that use ROS and the Jetson family, so I hope this is only the beginning of my experience with these.

References

[1] Cho, Y. and Hwang, J.Y., 2015. A study on EKF-SLAM simulation of autonomous flight control of quadcopter. International Journal of Software Engineering and Its Applications, 9(9), pp.269-282.

[2] García, S., López, M.E., Barea, R., Bergasa, L.M., Gómez, A. and Molinos, E.J., 2016, May. Indoor SLAM for micro aerial vehicles control using monocular camera and sensor fusion. In 2016 international conference on autonomous robot systems and competitions (ICARSC) (pp. 205-210). IEEE.

[3] Gadde, C.S., Gadde, M.S., Mohanty, N. and Sundaram, S., 2021, July. Fast Obstacle Avoidance Motion in Small Quadcopter operation in a Cluttered Environment. In 2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT) (pp. 1-6). IEEE.

[4] Dobrea, D.-M. and Dobrea, M.-C. (no date) 'An autonomous UAV system for video monitoring of the quarantine zones', p. 15.

[5] Benson, J. (2022) Jetson Nano Hacks. Available at: https://jetsonhacks.com/author/kangalow/ (Accessed: 06 05 2022)

[6] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. 2015. /emphYou Only Look Once: Unified, Real-Time Object Detection. arXiv [cs.CV]. http://arxiv.org/abs/1506.02640.

[7] Hoiem, D., Chodpathumwan, Y., and Dai, Q. 2012. Diagnosing Error in Object Detectors. Computer Vision – ECCV 2012, Springer Berlin Heidelberg, 340–353.

[8] Teed, Z. and Deng, J. (2020) 'RAFT: Recurrent All-Pairs Field Transforms for Optical Flow', arXiv:2003.12039 [cs] [Preprint]. Available at: http://arxiv.org/abs/2003.12039 (Accessed: 27 February 2022).