Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022
doi:10.20944/preprints202202.0076.v1

*Article*

# Human-Vehicle Leader-Follower Control using Deep Learning-Driven Gesture Recognition

**Joseph Schulte [1], Mark Kocherovsky [2] and Nicholas Paul[3] and Mitchell Pleune[4] and Chan-Jin Chung [5]**

1 Lawrence Technological University; jschulte@ltu.edu
2 Lawrence Technological University; mkocherov@ltu.edu
3 Lawrence Technological University; npaul@ltu.edu
4 Lawrence Technological University; mpleune@ltu.edu
5 Lawrence Technological University; cchung@ltu.edu

**Abstract:** Leader-follower autonomy (LFA) systems have so far only focused on vehicles following other vehicles. Though there have been several decades of research into this topic, there has not yet been any work on human-vehicle leader-follower systems. We present a system in which an autonomous vehicle — our ACTor 1 platform — can follow a human leader who controls the vehicle through gestures. We successfully developed a modular pipeline that uses artificial intelligence/deep learning to recognize hand-and-body gestures from a user in view of the vehicle's camera and translate those gestures into physical action by the vehicle. We demonstrate our work using our ACTor 1 platform, a modified Polaris Gem 2 electric vehicle. This work has numerous applications, such as material transport in industrial contexts.

**Keywords:** leader-follower; autonomous vehicles; self-driving car; machine learning; neural networks; deep learning; YOLO; posenet; pose estimation; gesture recognition

## 1. Introduction

Leader-follower autonomy (LFA) systems, whereby one or more autonomous vehicles can follow other vehicles without the need for a human operator, is a field that has seen continuous development over the last several decades. Studies of LF systems include the development of mathematical models [1–5], testing in simulations [1–8], and live experiments [4–6,8–10] with both two-robot and multi-robot systems. Demonstrations of LF systems have been conducted with real applications in mind on land, air, and sea [6,10]. The use of neural networks has also been introduced as a substitute for traditional mathematical pathfinding [9].

Despite the work done in vehicle-vehicle (VV) systems, academic LF development has largely ignored *Human*-Vehicle (HV) systems on large-scale autonomous vehicles. Though there has been study on human-robot following [11,12] as well as application in the commercial sector [13], these studies are more concerned with smaller robots and personal interactions rather than medium or large-sized vehicles.

We have already demonstrated that practical and reliable autonomous human-vehicle leader-follower behavior is possible [14] using our test vehicle, known as ACTor 1 (Autonomous Campus Transport 1, Figure 1). The ACTor is a Polaris Gem 2 modified with sensory capability, including a LIDAR and several cameras, and computer control through a DataSpeed drive-by-wire system. We use the Robot Operating System (ROS) to program interactions between the vehicle, its sensors, and user commands [15,16]. ROS is a software platform that abstracts the program-level control of the vehicle's hardware for standardization. If a manufacturer chooses to provide ROS support for their product, an engineer can integrate that piece of hardware into their control program without needing to know that hardware's proprietary control architecture [17]. The previous demonstration used traditional computer vision processing to identify ArUco

© 2022 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1

**Figure 1.** An image of ACTor 1 [16].

markers, a type of fiducial marker similar to QR codes that are optimized for reliable recognition and location in varied circumstances, and associate them with a human "leader" [14,18]. Object and human detection was accomplished with the YOLOv3 object detection system [19]. The human leader can then walk around, and ACTor 1 will follow them until either the follow function is disabled or the human leader walks out of camera view.

In this study, we demonstrate the practical application of deep-learning based gesture recognition as a control mechanism for human-vehicle LFA as a more natural and versatile alternative to traditional fiducial markers. We also demonstrate that gesture recognition can be used to control a vehicle in a real-world scenario. We envision several useful applications for such HV systems beyond automobile control, such as material transport in loading bays, construction sites, and factory floors.

In Section 2 we begin by explaining the fundamentals of neural networks, the gestures chosen for control, and our gesture recognition system. We continue by introducing the ACTor 1 platform, briefly outlining the basic principles of ROS and explaining our ROS program architecture and implementation on ACTor 1 in Section 3. In Section 4 we describe our live demonstrations and their results. Finally, in Section 5, we discuss our results and avenues of future study.

## 2. Gesture Recognition

### *2.1. Neural Network Fundamentals*

Artificial neural networks (NNs) are algorithmic structures designed to perform machine learning, whereby a computer program can be "trained" on a dataset in order to "learn" to make useful predictions. Neural networks are made up of layers of logically connected nodes called **neurons**. A densely connected neural network (DNN) has an initial layer of **input neurons**, which represent the input data, a final layer of **output neurons**, which represent the result of processing, and layers of **hidden neurons** in between, where processing occurs. Each neuron takes in an input from each neuron in the previous layer (or themselves represent the input data in the case of the input neurons) and returns an output which is fed to each neuron in the next layer as input, except in the case of the output layer, which simply returns the result(s), also called a **prediction**.

Each connection is associated with a **weight value** $w$. Each layer also is given a **bias value** $b$. During processing, each sample vector is multiplied by the matrix of weights, to which the biases are added, and an **activation function** is applied. The activation function is used to constrain the output values as they propagate through the network, which prevents highly unbalanced results. Choosing the correct activation function

depends on the problem at hand, but in our system (see Section 2.3.2), we use rectified linear units (ReLu) ($f(x) = \texttt{max}(x, 0)$) for our hidden layers. In **this project, we have a multiclass classification problem**, and thus make use of a **softmax** function for the output layer, which returns a probability distribution for each class. Since we have three potential classes, then the `softmax` function will return three values for each data sample; the highest value is in the position of the most likely class. If the sample likely belongs to the second class, then that highest probability value will be in position `1`.

The output of each neuron in layer $l_b$, which precedes layer $l_a$ and is followed by layer $l_c$ is given as follows:

$$y = f(\Sigma_{i=0}^{n} x_i w_i + b) \tag{1}$$

where $y$ is the output of the neuron, $n$ is the amount of connections to the given neuron in $l_b$, $x$ represents the input to $l_b$ (so the output of each neuron in $l_a$), $w$ represents the weights associated with the connection to each neuron in $l_a$, $i$ represents the value in a given vector in position $i$, $f(x)$ applies the chosen activation function, and $b$ represents the bias value. $y$ is then fed forward to each neuron in $l_c$.

During training, the DNN is shown each sample or batch (subset) of samples in the training dataset and their corresponding target result (called the **label**) and the resulting predictions are recorded. A **loss value** is then computed to measure the error between the target label and the prediction. **Backpropagation** is then applied to adjust each weight value accordingly. This is repeated several times; each loop through the dataset is called an **epoch**. If there is one epoch, then the DNN will see each input once. If there are two epochs, it will see each input twice, and so forth.

The designer may also arrange for a subset of the input data to act as **validation** during training. In-between each epoch, the model is run through the validation dataset to test its effectiveness on unseen data, and the validation loss and accuracy are reported. During the training stage, **the engineer's goal is to minimize validation loss**. Otherwise, **overfitting** may occur, where the model performs *too* well on the training data, making it less responsive to unseen data. In our modular system, we use **categorical crossentropy** as our loss value, defined as

$$L = -\Sigma_{i=1}^{n} y_i \log_2 \hat{y} \tag{2}$$

where $L$ is the loss value, $n$ is the amount of classes, $y$ is the target value, and $\hat{y}$ is the predicted value [20–22].

#### 2.1.1. Convolutional Neural Networks

A common application of DNNs for image recognition is the **convolutional neural network** (CNN), which uses filtering layers. In training, these layers take every possible square of pixels of a given size (such as a 3x3 or 5x5 filter) and check them in every possible position and return a value corresponding to the percentage of matched pixels. These can then be condensed by **pooling**, which will take regions of these values (such as a 2x2 region) and create a smaller matrix with the `max` of the values in the region. The result of the pooling step is then fed to the next convolutional layer, and so forth until the image has been converted into a matrix of values with a size that the designer deems sufficient. The data from this matrix is then fed into dense layers to produce a final classification.

This essentially means that the network will learn to recognize abstract features that might be found on the image. For example, a CNN trained for facial recognition might learn the abstract representation of the human nose, eyes, and ears as separate filters rather than focus on the entire image at once [20].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1



*2.2. Gestures*

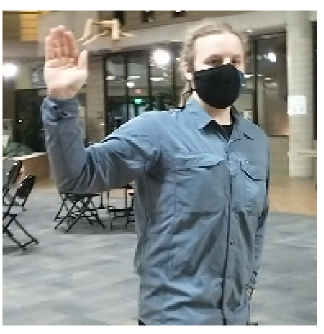
To control the ACTor, we decided on a set of two gestures, each corresponding to a command. A list of gestures and their corresponding function can be seen in Table 1.

| Gesture | Command |
|---|---|
| Hand on Heart | Begin Following |
| Palm out to the Side | Stop Following |
| Neither | No Change |

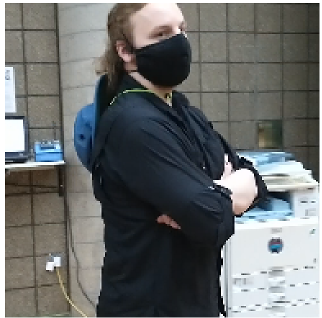
Table 1: List of visual gestures and their corresponding functions.

The `follow` command is indicated by the user putting their hand on their chest. Upon receiving a `follow` command, the system is instructed to recognize and track the user giving the command (now called the **target**). The system then maneuvers the ACTor to follow the user, but also works with the ACTor's LIDAR (see Section 4 for more information of the ACTor's systems) to remain a safe distance away from the target. Upon receiving a `stop` command, where the target puts their palm out to the side next to them, the system is instructed to stop following the user. If it detects neither pose, then it will continue behaving as per the previous instruction. Figure 2 depicts author J. Schulte performing each of the three pose cases. Our current dataset[1] includes 1959 `follow` images, 1789 `stop` images, and 1795 `none` images, though when training, the program selects a random sample of 1789 images from each subset to ensure training balance.

**(a)** Command to `follow` the user and label them as the target.



**(b)** Command to `stop` following the user



**(c)** No Command: follow previous command

**Figure 2.** Each of the three poses depicted by author J. Schulte.

*2.3. Neural Network Development*

We have developed a pipeline that efficiently translates camera video (structured as a stream of frames) into a command through object detection and pose estimation, but we built a more conventional CNN initially to see if it could be used. In this section, we describe our CNN construction, training, and results, and then explain our final modular pipeline and how it compares with the CNN. Both models were trained and tested on the same datasets, as well as under "lab conditions", i.e. with our laboratory as a background.

2.3.1. Building a Convolutional Neural Network

Before developing our current process, we built and tested a CNN using Keras, a leading library for neural network design in Python [23]. We also used TensorFlow, a Python library that provides tools for engineers to build machine learning programs for

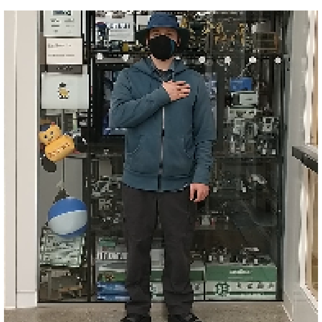
[1] Our dataset is publically available here: https://www.kaggle.com/dashlambda/ltu-actor-gesture-training-images. We ask that you cite this paper if you intend to use the images for any purpose.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1

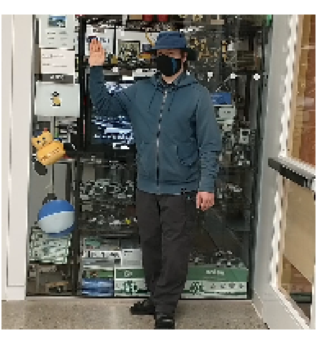


a variety of systems and applications. TensorFlow acts as an interface for Keras to enable smoother development and deployment of machine learning systems [24]. A diagram of our constructed CNN can be found in Figure A13, and a graph of the CNN's training can be seen in Figure 4. Note that **only the best model** (smallest validation loss value) **is saved during training** and that the **model would stop training if the validation loss did not improve for five consecutive epochs**. As loss decreases, accuracy tends to increase as the model learns. Due to the training settings, the model is only saved when validation loss improves during training.

Our basic CNN also failed to properly identify our gestures under laboratory conditions (using a live webcam feed in our laboratory) because differences of camera angles, backgrounds, lighting, people, and even clothing between pictures can potentially confuse a CNN if it is not trained on an large variety of conditions. To quantify this, we took 420 pictures of two of the authors in more diverse environments and ran the CNN on the new set, which can be seen in Figure 3. This returned a **test accuracy** of 26.84%, which is extremely low, with a **test loss** of 5.2013, which is very high. Thus, we concluded that the CNN model is not at all sufficient for gesture recognition.



**(a)** Command to `follow` the user and label them as the target.



**(b)** Command to `stop` following the user

**(c)** No Command: follow previous command

**Figure 3.** Examples of the testing dataset. When compared to the pictures in Figure 2, it is easy to see that the backgrounds are very different and author J. Schulte is wearing different clothes. This easily confuses basic CNN models but does not confuse our modular design.

The use of a CNN for classification would have also impeded our system in the long-term: first, we had to determine the position of the gesture in the image, which is possible to add to the program but not feasible in terms of development time. Second, our limited dataset, which consists solely of pictures of one of the authors, introduces the problem of **bias**. If an engineer is not careful enough and allows their training data to be biased, a machine-learning system — especially a neural network model — might learn to replicate prejudices shown by humans, even if this was not intended by the designer. For example, a facial recognition system that is poorly trained on images of non-Caucasian people might misclassify people with darker skin more often, or an automated hiring software might learn to prefer hiring men over women [25]. Since our dataset consists of a single person, using a CNN for gesture classification introduces an extreme bias in the program's execution. For these reasons, we chose an alternate method for gesture recognition.

#### 2.3.2. Modular Pipeline Design

To overcome the shortcomings of convolutional neural networks, we have built a modular pipeline with software reuse and flexibility in mind. We use a combination of complex prebuilt components and comparatively simple components of our own construction to efficiently translate a camera frame into commands for the vehicle. This design paradigm enabled us to develop our system quickly and without major software

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1



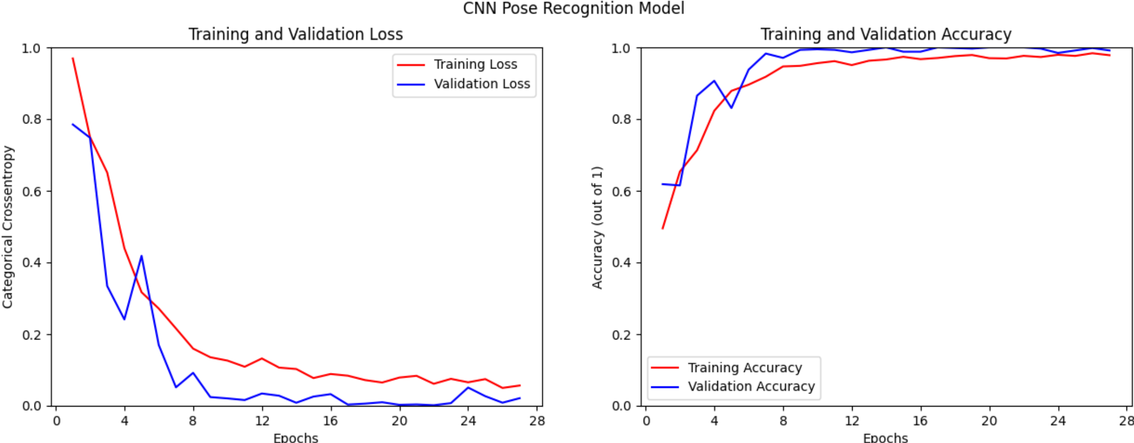



**Figure 4.** Graphs of the the CNN model's training process. On the left, the categorical crossentropy values for the training and validation sets are plotted by epoch. On the right, training and validation accuracy (proportion of correctly predicted labels) are plotted by epoch.

difficulties, since the most complex parts of the recognition module are prebuilt. A diagram of our pipeline is shown in Figure 6.

We decided to use a TensorFlow pose estimation model (posenet) that is readily available for single-pose (one person) estimation[2]. This model takes in an image of size 192x192 pixels and returns the estimated positions (x and y values) and confidence scores (how sure the model is that it has marked the point correctly) of 17 points on the body. [24,26].

To actually classify the pose, we built a simple DNN. It has as an input layer with 51 neurons (as each pose estimation contains 51 points), one hidden layer with 64 neurons, and an output layer with three neurons (one for each command). A diagram can be seen in Figure 5.

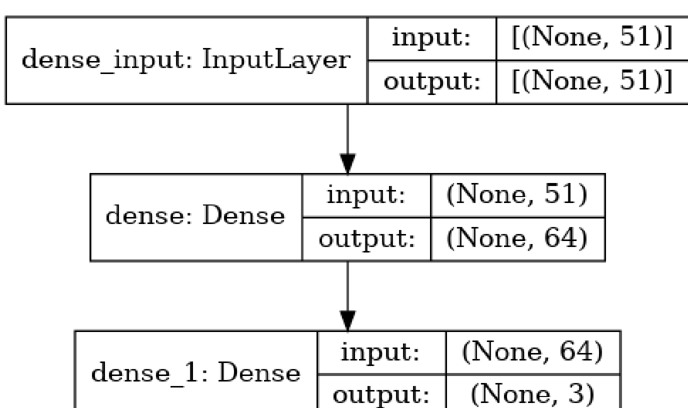



**Figure 5.** Automatically generated diagram of our classification NN. The first column is formatted as `layer_name: LayerType` and describes what kind of layer it is. The second and third columns describe the input and output of each layer. `None` indicates that the amount of inputs or outputs might be variable, but the number associated indicates how many values each input or output should/will have.

A diagram of the modular pipeline's classifier training process can be seen in Figure 7. Like in the CNN model, the training process was interrupted after the validation

[2] Available here: https://tfhub.dev/google/movenet/singlepose/lightning/4

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1



loss did not improve after five consecutive epochs, and only the model with the least validation loss was saved during training.

To generate the input for our gesture recognition model, we start with the object detection results from our pre-existing YOLO-based LFA architecture [19]. YOLO ouputs the edges of a bounding box around each person in the image, which we use to crop out a square frame focused on the person for each detection. We then resize this frame to the 192x192 target resolution. This is fed into the posenet, which returns the pose estimations. These values are then fed into the classifier, which returns the predicted command. The command is then sent to the vehicle (see Section 3).

This pipeline is advantageous because it separates the tasks of image recognition and gesture recognition. Image recognition and pose estimation is offloaded to robust pre-trained networks, which means we can use limited gesture datasets without introducing visual bias. Studies of TensorFlow Posenet or YOLOv3's biases in image recognition are outside the scope of this work.

Our process results in high performance under lab conditions (and as described in Section 3, strong performance in live testing). We also ran it on the same testing dataset that we used to evaluate our CNN, and found that it had a test accuracy of 0.8500 and a test loss of 0.4010. Both results are compiled in Table 2. As the table shows, since the modular design had a far lower test loss and far higher test accuracy than the convolutional design, the modular design is by far better than the CNN. Finally, an image of our pipeline's operation in experimental conditions are shown in Figure 8. The figure demonstrates that our pipeline is able to correctly classify gestures in a real-world scenario.

| Model | Test Loss | Test Accuracy |
|---|---|---|
| CNN | 5.2013 | 0.2684 |
| Modular | 0.4010 | 0.8500 |

Table 2: Comparison of CNN and Modular Designs when evaluated on the same testing dataset. Higher accuracy values and lower loss values indicate better models.

## 3. Vehicle Implementation

### *3.1. ACTor 1 Overview*

The ACTor 1 (Figure 1) is a Polaris Gem 2 modified with a suite of sensors and computer control. ACTor 1 can be controlled using any one of three computing units: An Intel NUC, a small off-the-shelf desktop computer equipped with an NVIDIA 1070Ti GPU, or a Raspberry PI 3B. These computers can interface with the drive-by-wire system developed by DataSpeed. The vehicle's sensors include a Velodyne VLP-16 360 degree 16 line 3D LIDAR, Allied Vision Mako G-319C camera with a 6mm 1stVision LE-MV2-0618-1 lens, and a Piksi Multi GNSS Module. The ACTor's systems are connected by Ethernet and/or CAN buses. A diagram of our hardware capabilities can be seen in Figure 9 [16].

### *3.2. ROS Fundamentals*

Our research group uses the Robot Operating System (ROS) to interface with the ACTor's hardware [15]. Despite the name, ROS is not an operating system nor does its use have to be confined to robots. ROS is a platform that provides an abstraction of all hardware with ROS support that allows an engineer to write control programs without needing to know the specifics of each piece of hardware.

ROS systems are structured as a set of implicitly-connected **nodes**. Each piece of hardware or component program has its own node through which it can (indirectly) interact with other nodes. For example, a motor would have a node that listens for movement commands and broadcast warnings if necessary, a sensor would have a node that would broadcast data and listen for commands to change detection parameters, and

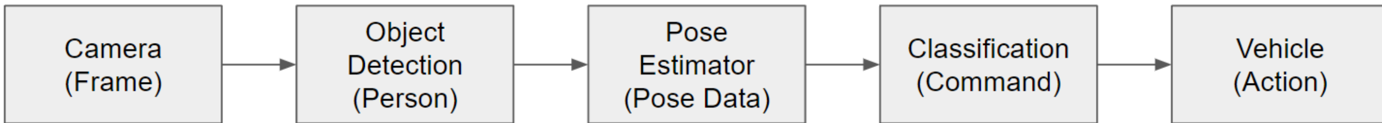



**Figure 6.** Our pipeline for gesture recognition and the resultant actions by the vehicle. This diagram is formatted as `Component (Returns)`. The camera returns a frame, the object detection takes the frame and returns a cropped photo of the person it is currently targeting, which the pose estimation uses to return the estimated pose data, which is in turn given to the classifier, which predicts a command, which is then sent to the vehicle, which finally performs an action.

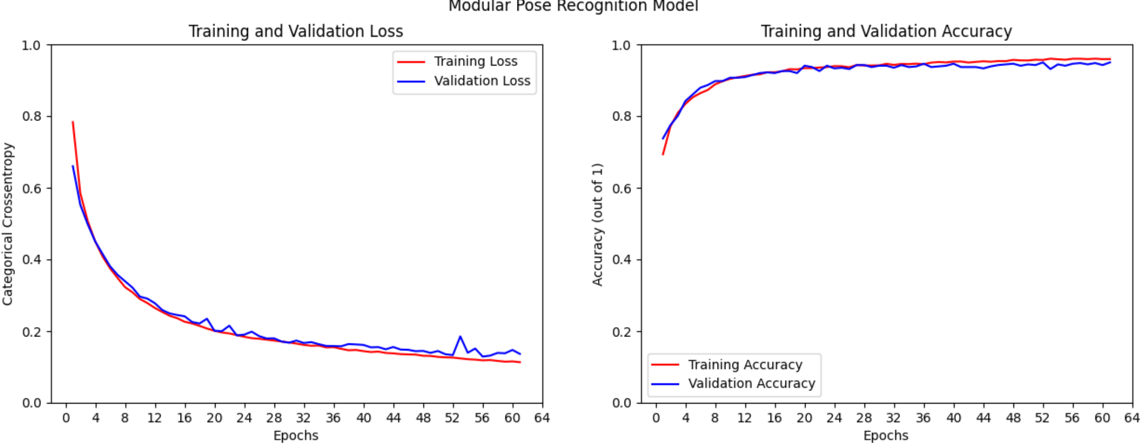



**Figure 7.** Graphs of the the modular classifier's training process. On the left, the categorical crossentropy values for the training and validation sets are plotted by epoch. On the right, training and validation accuracy are plotted by epoch. This classifier is the component in the fourth box ("classification") in Figure 6.

the control interface would have a third node that would listen for input from the two hardware components and broadcast output accordingly. The name of each node is set either by the engineer or the hardware manufacturers.

One of the main advantages of ROS is that the nodes are not explicitly connected. A node that wishes to broadcast output does not send that output to any specific node but instead broadcasts it on a logical channel called a **topic**. The act of broadcasting to the topic is called **publishing**. Which nodes are receiving the message, if any, are of no concern to the **publisher**. Nodes listen for messages on topics, which is called **subscribing**. Like the publishers, which nodes are publishing to a topic are of no concern to the **subscribers**. Nodes can publish or subscribe to as many topics as necessary. Like the node names, topic names can either be set by the engineers developing and using the system or the hardware manufacturer.

Topics have specific **message** types. Engineers and manufacturers may use message types that come with the library or its packages, such as integers, time information, `Twist` movement commands, and so forth. If these types are not sufficient, then an engineer may easily create their own message types.

Finally, ROS allows users to group programs into **launch files**. Launch files essentially contain the programs that need to be run for a system to operate, along with all necessary parameter assignments (a user may also re-assign parameters when starting the program via command line). Using the **dynamic reconfigure** package tools, users may also redefine parameters during execution. [17,27].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1

**Figure 8.** Our modular system performing gesture recognition on author M. Kocherovsky. YOLO determines the location of the author in the frame, which generates the pink bounding box surrounding him. The posenet component determines the location of the main joints and other key features on the author's body, which are displayed as red squares. Finally, the gesture recognition module has correctly identified the author's gesture as `follow`, and has marked him as the `Pose Target`, meaning that the vehicle will follow his movements.

### *3.3. ROS Node Design*

We have nine nodes running concurrently; a full diagram of our ROS nodes[3] can be seen in Figure 10.

1. `Velodyne Nodelet Manager`: This node provides an interface from our control unit to the LIDAR. It publishes the LIDAR sensor data for each point on the `Velodyne Points` topic.
2. `Mono Camera`: This node provides an interface from our control unit to the camera. It publishes the camera frames on the `Image Raw` topic.
3. `LIDAR Reporter`: This node receives raw input from the `Velodyne Points` topic, packages into a convenient format, and and publishes the reformatted data on the `LIDAR Points` topic.
4. `Darknet ROS`: This node subscribes to the `Camera` topic and runs object detection on camera frames using YOLO. It then publishes the location of each detected object in the image on the `Bounding Boxes` topic. This node was developed by Joseph Redmon, who also made YOLO [19,28].
5. `Detection Reporter`: This node subscribes to the `Bounding Boxes`, `LIDAR Points` and `Image Raw` topics and integrates their data to produce a coherent stream of information. It identifies the human detections reported by YOLO, superimposes their location in the image onto the 3D LIDAR point cloud to find their true location in three dimensions, idntifies targets based on the given criteria, and attempts to keep track of the target from frame to frame. It publishes the consoidated information to the `Detects Firstpass` topic.
6. `Detection Image Viewer`: This node subscribes to the `LIDAR Points` and `Detects` topics and to produce a visualization of the system's state. For each detection in

[3] Our code is also available here https://github.com/jschulte-ltu/ACTor_Person_Following. We ask that you cite this paper if you intend to use the code for any purpose.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1



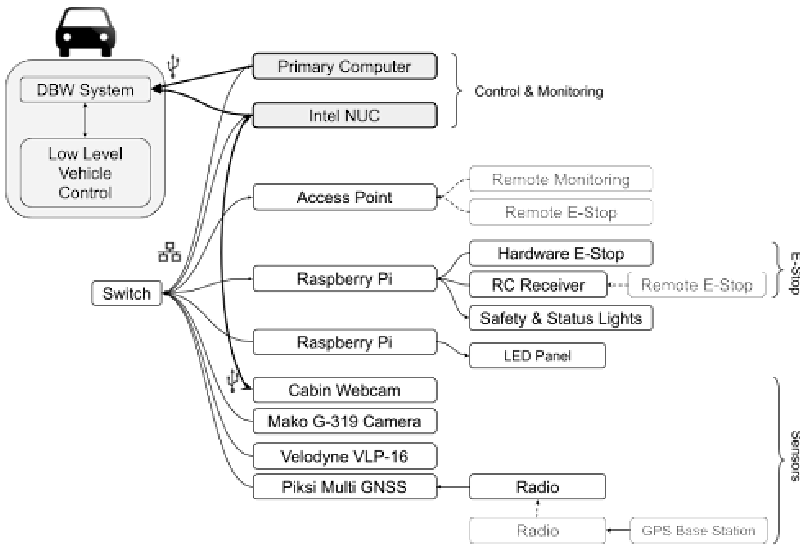



**Figure 9.** An image of the ACTor 1 hardware setup [16].

the image it draws the bounding box given by YOLO, draws the 17 pose points, and writes their distance from the vehicle's LIDAR system, the guesture they are performing, and whether or not they are a pose target. It can also superimpose the LIDAR point cloud onto the image and report the current action being performed by the vehicle. This node is purely for monitoring and visualization.

7. `Gesture Injection`: This node subscribes to the `Detects Firstpass` topic, implements our gesture recognition pipeline as described in Section 2.3.2 to identify each target's gesture and the corresponding commands, then republishes the detection information with these new identifications to the `Gesture Detects` topic. This node serves as a convenient and effective way to splice in the gesture detection pipeline with minimal alterations to our existing code.
8. `LFA` (Leader Follower Autonomy) `Controller`: This node subscribes to the `Detects`, `Follower Start` and `Follower Stop` topics and publishes to the `Display Message`, `Display RGB`, `Enable Vehicle`, and ULC command topics. This is the last node in our LFA pipeline, which takes the detection and gesture information generated by the prior nodes and determines the actual commands sent to the vehicle. Those commands are published on the `Command Velocity` topic.
9. `ULC` (Universal Lat-Long Controller): This node provides an interface between our control unit and the drive-by-wire system. It takes the command from the `Command Velocity` topic and translates them into action by the vehicle.

## 4. Experiment and Results

We ran our experiments on ACTor 1 using its main computing unit, equipped with an AMD Ryzen 7 2700 Processor and a ZOTAC GeForce GTX 1070 Ti GPU [16]. Our testing area was the atrium at Lawrence Technological University. The atrium is an open space where we have room to maneuver the vehicle. Our experiments were relatively simple; with the system running, a user would stand in front of ACTor 1, give a command to start, and then change position to see the ACTor's reaction. Our team can report that the system is capable of following a user when they give the `follow` command, pause when the ACTor is too close to the user (using the LIDAR), and stop when the user either gives the `stop` command or is out of the camera's field of view. Figures 11 and 12 demonstrate our system in operation[4].

[4] A video demonstration is available on YouTube at the following link: https://www.youtube.com/watch?v=A_CExLAEiB0

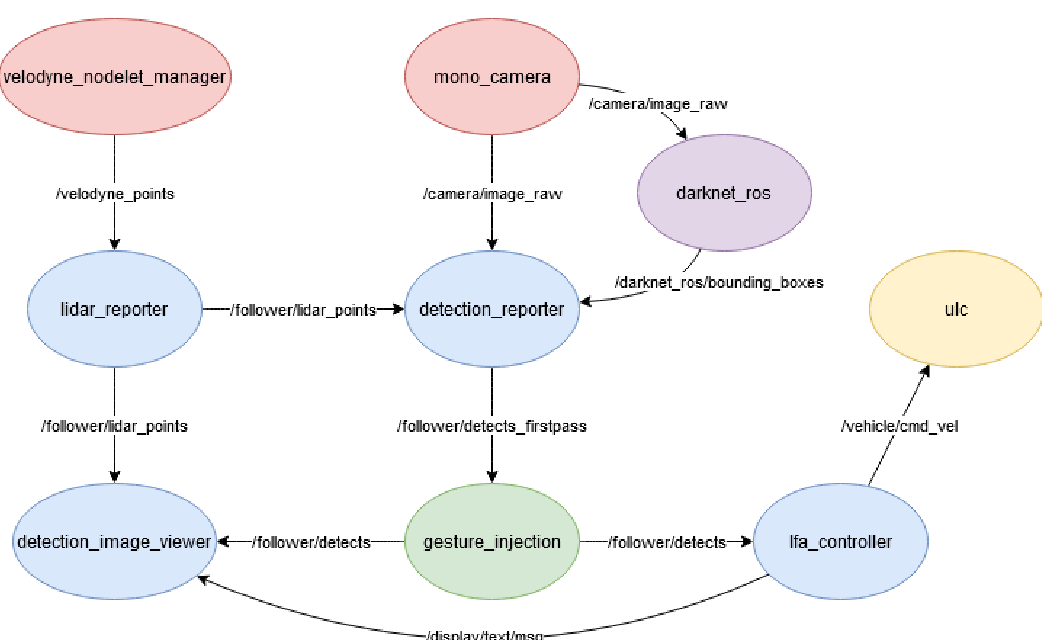



**Figure 10.** A Diagram of the main ROS Nodes (ellipses) and topics (arrows) specific to our leader follower system. Nodes marked in red indicate hardware nodes which publish sensor data. Blue nodes represent helper nodes. The `Darknet ROS` node manages object detection, `Gesture Injection` runs the pose estimation and classification, and `ULC` manages interactions with the vehicle's drive-by-wire system.

## 5. Discussion

### 5.1. Summary

We have demonstrated that practical and reliable autonomous human-vehicle leader-follower systems are possible and feasible with current technology. We have demonstrated that it is possible to use pre-trained neural network models, namely YOLO, and newly trained DNN classifiers to recognize gesture commands in dynamic environments and translate those commands into actions on an integrated self-driving car computer platform through a pipeline architecture. We use modular software design and software reuse principles to accelerate development, improve quality, and limit technical difficulties.

### 5.2. Future Work

We envision four areas of future development and study. From a software perspective, we wish to further optimize our software to achieve faster processing speeds, thus making the system more responsive. We also wish to improve obstacle detection and path planning to make the system safer and more versatile. In addition, we seek to add more gestures to add additional functionality to the system. Finally, we believe that tests should be run in more realistic environments and with potential applications in mind, such as material transport in manufacturing, construction, and other industrial contexts.

**Author Contributions:** Conceptualization, C.C., J.S, M.K. M.P., and N.P.; design methodology, J.S. and M.K.; software implementation, J.S. and M.K.; validation & testing, J.S. and M.K.; data curation, J.S, and M.K.; writing—original draft preparation, M.K.; writing—review and editing, C.C.; visualization, M.K. and J.S.; project administration, C.C.; funding & acquisition, C.C; Vehicle setup maintenance: M.P. and N.P.; Vehicle troubleshooting: M.P. and N.P.; All authors have read and agreed to the published version of the manuscript.

**Funding:** This specific research received no external funding. However the following companies and organizations sponsored the acquisition of ACTor electric vehicles, sensors, on-board computers, by-wire systems, other parts, and vehicle maintenance cost: US Army Ground Vehicle

*Preprints* (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1

**Figure 11.** Author J. Schulte gives ACTor 1 the `follow` command. Author M. Kocherovsky sits in the driver's seat as a safety precaution. Picture taken by author CJ Chung.

**Figure 12.** ACTor 1 autonomously follows author M. Kocherovsky in the atrium of Lawrence Technological University. Author J. Schulte sits in the driver's seat as a safety precaution. Picture taken by author M. Pleune.

Systems Center, DENSO, SoarTech, Realtime Technologies, GLS&T, Hyundai MOBIS, Dataspeed, NDIA Michigan, Veoneer, and Lawrence Technological University.

**Acknowledgments:** The authors thank the Department of Mathematics and Computer Science and the College of Arts and Sciences at Lawrence Technological University for administrative and financial support. The authors also thank ACTor team members Thomas Brefeld, Giuseppe DeRose, Justin Dombecki, and James Golding for their technical support and assistance.

**Conflicts of Interest:** The authors declare no conflict of interest.

1. Consolini, L.; Morbidi, F.; Prattichizzo, D.; Tosques, M. Leader–follower formation control of nonholonomic mobile robots with input constraints. *Automatica* **2008**, *44*, 1343–1349. doi: https://doi.org/10.1016/j.automatica.2007.09.019.
2. Cheok, K.; Iyengar, K.; Oweis, S. Smooth Trajectory Planning for Autonomous Leader-Follower Robots. Proceedings of 34th International Conference on Computers and Their Applications; Lee, G.; Jin, Y., Eds. EasyChair, 2019, Vol. 58, *EPiC Series in Computing*, pp. 301–309. doi:10.29007/n6kt.
3. Ramadan, A.; Hussein, A.; Shehata, O.; Garcia, F.; Tadjine, H.; Matthes, E. Dynamics Platooning Model and Protocols for Self-Driving Vehicles. 2019 IEEE Intelligent Vehicles Symposium, 2019, pp. 1974–1980. doi:10.1109/IVS.2019.8813864.

*Preprints* (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1




4. Ng, T.C.; Guzman, J.; Adams, M. Autonomous vehicle-following systems : a virtual trailer link model. 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005, pp. 3057–3062. doi:10.1109/IROS.2005.1545427.
5. Kehtarnavaz, N.; Griswold, N.; Lee, J. Visual control of an autonomous vehicle (BART)-the vehicle-following problem. *IEEE Transactions on Vehicular Technology* **1991**, *40*, 654–662. doi: 10.1109/25.97520.
6. Simonsen, A.S.; Ruud, E.L.M. The Application of a Flexible Leader-Follower Control Algorithm to Different Mobile Autonomous Robots. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020, pp. 11561–11566. doi: 10.1109/IROS45743.2020.9341780.
7. Lakshmanan, S.; Yan, Y.; Baek, S.; Alghodhaifi, H. Modeling and simulation of leader-follower autonomous vehicles: environment effects. Unmanned Systems Technology XXI; Shoemaker, C.M.; Nguyen, H.G.; Muench, P.L., Eds. International Society for Optics and Photonics, SPIE, 2019, Vol. 11021, pp. 116 – 123.
8. Solot, A.; Ferlini, A. Leader-Follower Formations on Real Terrestrial Robots. Proceedings of the ACM SIGCOMM 2019 Workshop on Mobile AirGround Edge Computing, Systems, Networks, and Applications; Association for Computing Machinery: New York, NY, USA, 2019; MAGESys'19, p. 15–21. doi:10.1145/3341568.3342107.
9. Kehtarnavaz, N.; Groswold, N.; Miller, K.; Lascoe, P. A transportable neural-network approach to autonomous vehicle following. *IEEE Transactions on Vehicular Technology* **1998**, *47*, 694–702. doi:10.1109/25.669106.
10. Tang, Q.; Cheng, Y.; Hu, X.; Chen, C.; Song, Y.; Qin, R. Evaluation Methodology of Leader-Follower Autonomous Vehicle System for Work Zone Maintenance. *Transportation Research Record* **2021**, *2675*, 107–119. doi:10.1177/0361198120985233.
11. Setiawan, S.; Yamaguchi, J.; Hyon, S.H.; Takanishi, A. Physical interaction between human and a bipedal humanoid robot-realization of human-follow walking. Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), 1999, Vol. 1, pp. 361–367 vol.1. doi:10.1109/ROBOT.1999.770005.
12. Morioka, K.; Lee, J.H.; Hashimoto, H. Human-following mobile robot in a distributed intelligent sensor network. *IEEE Transactions on Industrial Electronics* **2004**, *51*, 229–237. doi: 10.1109/TIE.2003.821894.
13. More info about Travelmate, 2021. Accessed: 2021-09-26.
14. Schulte, J.; Dombecki, J.; Pleune, M.; DeRose, G.; Paul, N.; Chung, C. Developing Leader Follower Autonomy for Self-driving Vehicles Using Computer Vision and Deep Learning. Lawrence Technological University Eighth Annual Research Day, 2021. Poster, Online.
15. Paul, N.; Pleune, M.; Chung, C.; Warrick, B.; Bleicher, S.; Faulkner, C. ACTor: A Practical, Modular, and Adaptable Autonomous Vehicle Research Platform. 2018 IEEE International Conference on Electro/Information Technology (EIT), 2018, pp. 0411–0414. doi: 10.1109/EIT.2018.8500202.
16. DeRose, G.; Brefeld, T.; Dombecki, J.; Pleune, M.; Schulte, J.; Paul, N.; Chung, C. ACTor: IGVC Self-Drive Design Report, 2021.
17. Stanford Artificial Intelligence Laboratory et al.. Robotic Operating System, 2018.
18. OpenCV: Detection of ArUco Markers. Online, 2021.
19. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**.
20. Chollet, F. *Deep learning with Python*, 1. ed.; Manning, 2017.
21. Dongare, A.; Kharde, R.; Kachare, A.D.; others. Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)* **2012**, *2*, 189–194.
22. Peltarion. Categorical crossentropy. Accessed: 2021-12-21.
23. Chollet, F.; others. Keras. https://keras.io, 2015.
24. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.
25. Caliskan, A.; Bryson, J.; Narayanan, A. Semantics derived automatically from language corpora contain human-like biases. *Science* **2017**, *356*, 183–186. doi:10.1126/science.aal4230.
26. Alphabet Inc.. movenet/singlepose/lightning, 2021.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1



27. O'Kane, J.M. *A gentle introduction to ROS*; Jason M. O'Kane, 2014.
28. Redmon, J. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/, 2013–2016.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 February 2022 doi:10.20944/preprints202202.0076.v1



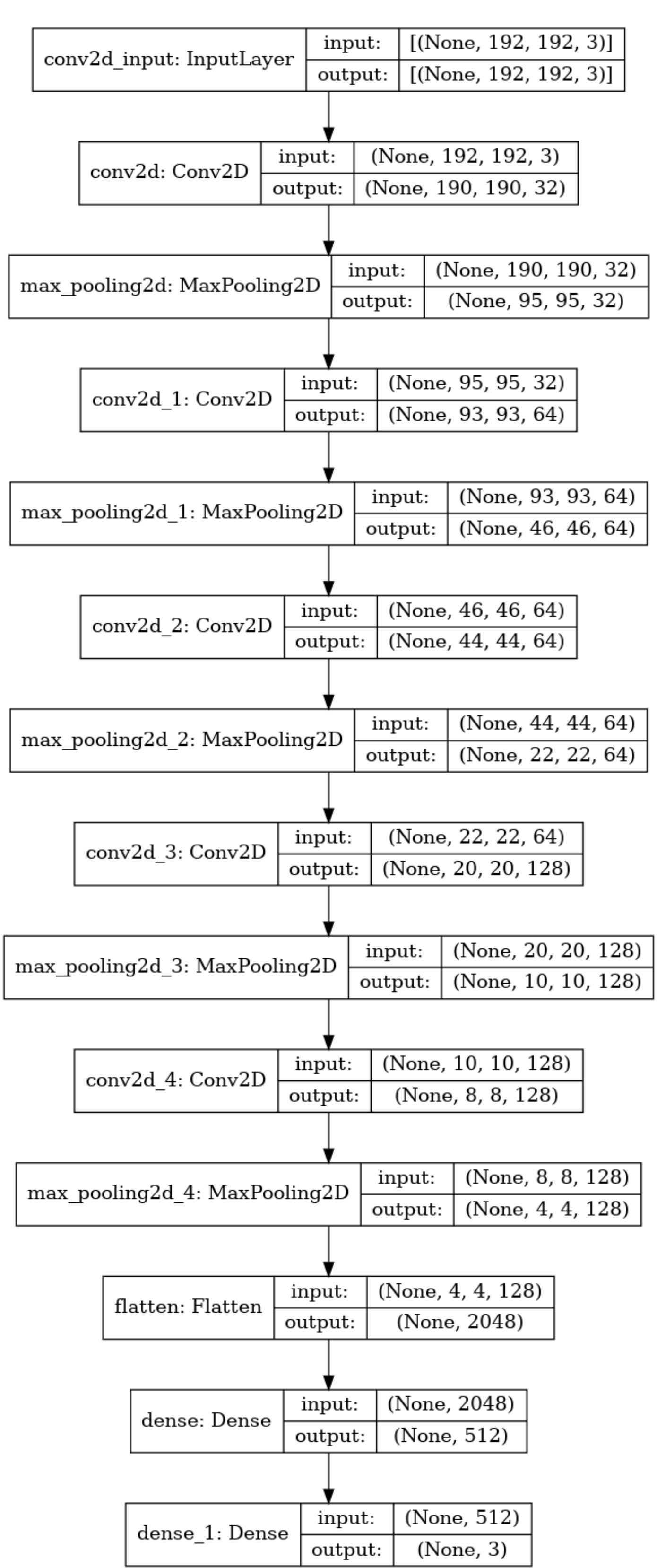



**Figure A13.** Diagram of our Convolutional Neural Network.