# Visual Navigation in Dynamic Environments

Son Tran
sontran@berkeley.edu

Zuyong Li
zuyong_li@berkeley.edu

Shawn Shacterman
shawnshact@berkeley.edu

Yuanrong Han
rexhanh123@berkeley.edu

***Abstract*—In this paper, we extend recently proposed solutions for tackling autonomous navigation in unknown environments. We demonstrate a method for identifying dynamic obstacles only using information from an RGB-D camera. Additionally, we extend current state-of-the-art methods for quickly generating simulated scenarios in which a robot must navigate to a goal while avoiding moving obstacles. Finally, we propose an approach for combining the identification of dynamic obstacles with these random scenarios to predict waypoints on the path from a start position to a goal position in novel environments.**

**https://robot-gang.github.io/visual-navigation/**

***Index Terms*—robotics, visual navigation, dynamic environments**

## I. Introduction

Autonomous robot navigation has the potential to enable many real-world applications, such as service robots that deliver food or medicine. In most of these applications, robots usually work in dynamic environments where humans or other objects are moving. Robots need to adjust their movements to avoid collision with other moving objects.

For example, when the robot observes an object moves from left to right in front of it, then it should pass that object on the left to safely avoid the collision.

Specifically, the methods for autonomous robot navigation among moving objects need to detect the objects, calculate or predict objects' motion, and react to them safely. Moving object detection is difficult because of illumination changes, background subtraction, and occlusion. Determining objects' motion is challenging because the objects' navigation goal is not known to the robot, and the objects can suddenly change their motion (speed, path, etc.). The robot's reactions to moving objects need to carefully balance reaching the robot's target and avoiding a collision. The mentioned reasons make navigating in dynamic environments difficult, especially when the robot is operating in unknown environments. Classical solutions for autonomous robot navigation usually involve specialized methods for perception, planning, and control. Although these solutions work well for many circumstances, it is hard to set up and often requires specialized environment-specific tuning. Therefore, it is hard to generalize classical solutions to novel environments.

Recently, end-to-end learning methods, which map sensor outputs directly to robot actions [1], [2], [3], [4], have shown promising results in increasing robustness to noise, and generalizing to novel environments. However, because these methods do not explicitly account for the dynamics of the vehicle/robot, the inputs generated in end-to-end learning navigation schemes are often jerky and suboptimal. Authors in [5] combine the benefits of classical and learning-based

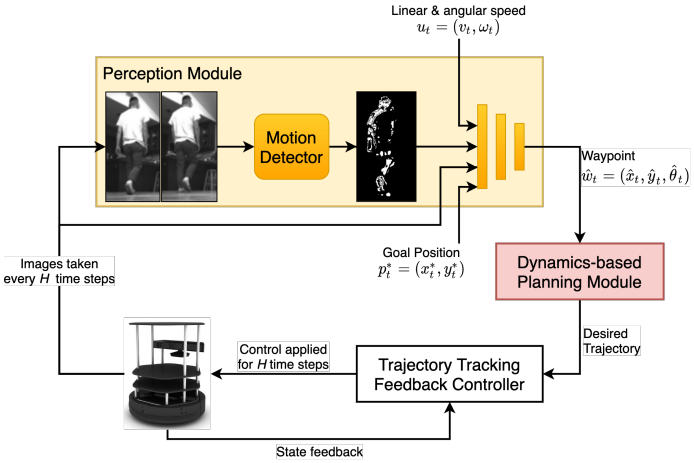


Figure 1. **Overview:** The framework to navigate in an unknown dynamic environment. This framework is based on the architecture from [5]. The framework consists of a learning-based perception module and a dynamics model-based planning module. The perception module predicts a waypoint based on the current first-person RGB-D images. This waypoint is used by the model-based planning module to design a feedback controller that smoothly regulates the system to the waypoint. This process is repeated until the robot reaches the goal.

methods for autonomous robot navigation. The authors use a learned perception module with a model-based controller for navigation in an unknown environment. This work leverages a learned Convolutional Neural Network (CNN) to predict a waypoint, the vehicle's next desired state, and uses optimal control to actually reach the waypoint. This method has been shown to execute smooth, efficient trajectories in novel, static environments (both simulation and real).

Authors in [6] use the architecture proposed in [5] but extend it to environments with moving humans. Specifically, the robot is navigating in an unknown space shared with a single human, whose trajectory is also unknown. To train such a method requires training data, which can be obtained either from real robots or from a simulation. While robot manipulation often trains from real data, it poses safety, privacy, and logistical challenges. At the same time, training in simulation poses a problem as it requires a simulation of visually realistic humans and their motion. For this reason, training navigational agents often rely either on synthetic humans devoid of low-level visual cues such as color, texture, and pose, or training on photorealistic static environments and evaluating in dynamic environments. To address challenges of training visual navigation agents around humans, authors in [6] introduce the Human Active Navigation Dataset (HumANav),

an active dataset for navigation around humans in photo-realistic environments. They use it to generate supervision for optimal waypoints for the method described in [5]. The HumANav dataset consists of scans of 6000 synthetic but realistic humans from the SURREAL dataset [7] placed in office buildings from Stanford Large Scale 3D Indoor Spaces Dataset [8]. HumANav allows for user manipulation of human agents within the building and provides photorealistic renderings (RGB, Depth, Surface Normals, etc.) via a standard perspective projection camera.

In this work, we use the architecture proposed in [5] and extend the work in [6] to general dynamic environments with many types of moving objects instead of only a single human. To achieve our goal, we need to modify the perception module of the framework in [5]. The overview of the framework that we used is present in Figure 1. Instead of taking one RGB image and predicting the waypoint from it, the new perception uses multiple RGB-D images, detects moving objects in the scene, and predicts a waypoint. We have developed the Algorithm 1 to detect moving objects in a scene, and modified the HumANav dataset in [6] to add multiple humans moving in different trajectories.

Overall, this paper introduces our approach for visual navigation, planning, and control in unknown dynamic environments. We also present some of our results for detecting moving objects and modifying the HumANav dataset in [6].

## II. Related Work

There is a large body of work that deals with autonomous navigation. Because our work primarily deals with navigation in environments with dynamic obstacles, this section will summarize related work with a similar focus. Our approach is heavily inspired by [5], which combines learning (for perception), spline-based motion planning, and optimal control to enable navigation from a starting position to a goal position in some fixed world frame. While the model in [5] assumes a static environment, it is able to navigate around a limited set of dynamic obstacles due to an MPC control loop that allows the system to periodically replan its trajectory to a different waypoint if the current one is no longer optimal/feasible. More recently, the authors of [5] released a new work, [6], that extends their navigation scheme to reliably work in scenarios where humans are modeled as dynamic obstacles. In this new navigation scheme, waypoints are generated by feeding an image, goal position, and current robot velocity into a neural network. While observing a single image in a navigation scheme is potentially applicable to humans (as the network may be able to infer motion from the analyzing the gait of the person), our method is designed to observe a sequence of images over a time period. Our hope is that the network can infer trajectories from a more general class of dynamic obstacles with potentially fewer indicators of motion. We believe that there is a precedent for this approach that has been set in prior reinforcement learning research. In [9], the authors found that they needed to feed multiple frames from the Atari game *Breakout* for their agent to observe the direction of the ball's motion and react accordingly. Our approach aims to leverage sequential information to properly observe the motion of obstacles in an unknown environment.

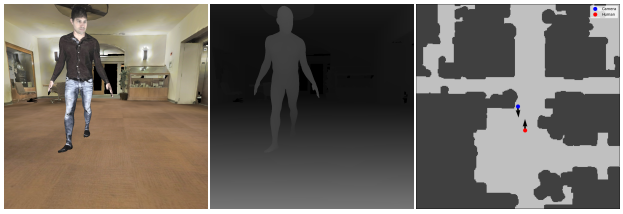

Figure 2. These figures are generated by HumANav using SURREAL and S3DIS dataset. On the left is the generated RGB images with synthetic human mesh, in the middle is the depth information, on the right is the map information

In addition to using image sequences, our approach aims to identify moving objects in the environment. There also exists a body of related work that deals with moving object detection. The key idea is to compare the feature points of different frames. The method of extracting and matching feature points is well-studied. One well known approach is to use Harris corner detection, adaptive non-maximal suppression to obtain nicely distributed feature points [10]. Another way to detect moving objects is to check whether the feature point in one frame is on the epipolar line in other frames and then remove all features that lie inside a semantic segmentation [11]. It requires the segmentation of the moving objects, so it is not suitable for dynamic environments with unknown obstacles. Our method does not require any semantic segmentation for motion detection.

## III. Methods

### A. Data Generation

In order to forego the challenges of creating and pre-processing custom meshes for our obstacles, we decided to use the HumANav dataset [6] to generate and manipulate randomized human meshes. HumANav combines the SURREAL (Synthetic hUmans foR REAL tasks) dataset [7] and SD3DIS (Stanford Large Scale 3d Indoor Spaces Dataset) dataset [8] to allow for easy injection of human obstacles into photorealistic 3D environments. The HumANav Dataset contains 6000 different human meshes with different poses, genders, body shapes, lighting conditions and velocities. Since we can control of the position and velocity of the human, we wrote a class to pick a random human position with a known speed and then using the known position and velocity to generate different human trajectories (straight line, circle, random-walk etc.). We set the total time and velocity of the moving human, so we can obtain the position of the human at the next time step and generate a sequence of images. This can be seen in the Figure 2, where the randomly generated human is moving in a straight line with a constant velocity. Previously, HumANav only supported placing one human in an environment at a time. By modifying the rendering engine, we can now render and manipulate multiple humans

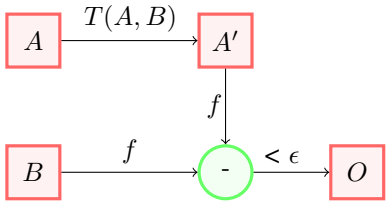


Figure 3. Scheme of finding changes in intensity and depth. $A$ and $B$ are inputs, *RGB* images or depth arrays. $T(A, B)$ transforms $A$ into $B$'s projection plane or camera frame. $f$ is a function applied to the data before comparison. The difference is then compared to a threshold $\epsilon$ to produce the output $O$

in an environment, which allows for a more diverse range of navigation scenarios for training our perception module.

Most importantly, HumANav keeps a realistic human texture and pose, which will be important for capturing an environment that the robot is likely to encounter in the real-world. When the images are rendered, different velocities will correspond to different poses (larger velocity corresponds to larger human step size).

### B. Moving Object Detection

The motion of an object is reflected in the change of intensity and depth. If an object is moving, the distance between the camera center and the object varies with respect to time. It will also cause intensity changes due to covering and uncovering of the background, and changes of camera pose. To detect the changes in intensity and depth, we used the scheme in Figure 3. $A$ and $B$ are the inputs, and $T(A, B)$ is the corresponding transformation. When detecting changes in intensity, $A$ and $B$ are the *RGB* images, and $T(A, B)$ is the *homography* that can be used to project $A$ into $A'$ viewed in $B$'s projection plane. When detecting changes in distance, $A$ and $B$ are the depth arrays, and $T(A, B)$ is the *rigid body transformation* of the camera pose which can be used to transform points encoded in $A$ to $A'$ viewed in the camera's local frame corresponding to $B$. Before passing $A'$ and $B$ to function $f$, a Gaussian filter is applied to reduce the effect of noise. $f$ is a function to normalize the distribution of $A'$ and $B$ to $\mathcal{N}(0, 1)$.

$$f(X) = \frac{X - \bar{X}}{\sigma_X} \tag{1}$$

In the equation 1, $\bar{X}$ and $\sigma_X$ are the mean and standard deviation of $X$ respectively. This normalization makes the comparison invariant to affine changes in intensity and depth [10]. The absolute difference is then compared to a threshold $\epsilon$ to produce the output $O$. The complete algorithm for moving object detection is presented in Algorithm 1.

*1) Feature Matching:* To compute the transformation $T(A, B)$, the key points were computed using OpenCV's feature extraction method. We used FAST detector and BRIEF descriptor to find features. To match these key points between two images, we used BFMatcher's $k$ nearest neighbors to match every key point in $A$ with two key points in $B$, and then apply ratio test to determine whether a match is good. If the key point $p_1$ in $A$ is matched to $q_1, q_2$ in $B$, then $(p_1, q_1)$ is a good match if $d_{p_1q_1} < 0.75 d_{p_1q_2}$, where $d(p_i, q_j)$ is the Hamming distance between $p_i$ and $q_j$.

**Algorithm 1:** Moving Object Detection

**Input:** images $A, B$, depth arrays $A_d, B_d$
extract and match features in $A$ and $B$;
detect changes in intensity $\Delta I$;
detect changes in depth $\Delta\Lambda$;
combination of changes $\Delta$;
**Output** a binary map of moving objects $\Delta < \epsilon$

*2) Detecting Changes in Intensity:* For robot moving at low speed, the camera pose does not change dramatically in a short time interval. Thus, we can warp one *RGB* image $A$ into $B$'s projection plane to obtain $A'$ by applying the homography matrix on the image $A$. The homography matrix $H$ is computed by using good matches that are produced from the *Feature Matching* step and the random sample consensus (RANSAC) method. The OpenCV's *findHomography* function takes in coordinates of matching key points and returns the homography matrix corresponding to these matches. To actually warp image $A$ into $B$'s projection plane, we copy the intensity value at point $p_A$ to the location $p_B = Hp_A$ generates $A'$, which is done by calling OpenCV's *warpPerspective* function. The changes in intensity is calculated by $\Delta I = |f(A') - f(B)|$.

*3) Detecting Changes in Depth:* The depth information changes when there is motion in the scene such as moving objects or camera pose changes. If the camera pose $(R, t)$ of $B$ relative to $A$ is known from the robot's state, then points in the local coordinates of $A$ can be re-projected into $B$'s camera frame using the following equations, assuming frame $A$ is the global frame.

$$\lambda_A x_A = K[I|0] \begin{bmatrix} X \\ 1 \end{bmatrix} = KX \tag{2}$$

In the equation 2, $K$ is the camera intrinsic matrix, $x_A$ is the pixel homogeneous coordinates of points in $A$, and $X$ is the global coordinates corresponding to $x_A$. The re-projected pixel coordinates $x'_A$ is given by

$$\lambda'_A x'_A = K[R|t] \begin{bmatrix} X \\ 1 \end{bmatrix} = K(RK^{-1} x_A \lambda_A + t) \tag{3}$$

The changes in depth for one point is then calculated by $\delta\lambda = |\lambda_B - \lambda'_A|$. We calculated the changes in depth for all points at once by substituting $x_A, x'_A, \lambda_A, \lambda'_A$ with matrices $\boldsymbol{x_A}, \boldsymbol{x'_A} \in \mathbb{R}^{3\times n}, \Lambda_A, \Lambda'_A \in diag(n)$. Thus the changes in depth for all point is then calculated by $\Delta\Lambda = |f(\Lambda'_A) - f(\Lambda_B)|$.

If the relative camera pose $(R, t)$ is unknown, it can be computed from the matching key points, shown in Figure 4. Let $(x_A, x_B)$ be a pair of matching key points in pixels which have the same global coordinate $\boldsymbol{X}$. First, we compute the local $3D$ coordinates of the feature points by combing the pixel coordinates and depth, $X_A = K^{-1} x_A \lambda_A, X_B = K^{-1} x_B \lambda_B$. The relation between $X_A$ and $X_B$ is $X_B = RX_A + t$. The

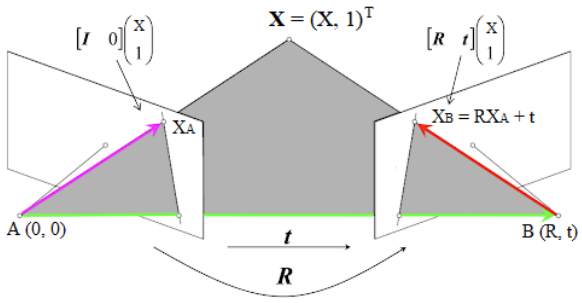


Figure 4. The same point $\boldsymbol{X}$ viewed in two camera frames $A$ and $B$. $X_A$ and $X_B$ are the local $3D$ coordinates of point $\boldsymbol{X}$ in the camera frame $A$ and $B$ respectively. $(R, t)$ is the camera pose for frame $B$, and frame $A$ is assumed to be the global frame.

local $3D$ coordinates of all matching key points can also be computed once by substituting $X_A, X_B, \lambda_A, \lambda_B$ with the corresponding matrices. Let $P$ denote all local points of $A$, and $Q$ denote all local points of $B$, then $Q = RP + t$, where

$$P = K^{-1} \begin{bmatrix} x_{A_1} & x_{A_2} & \dots \\ 1 & 1 & \dots \end{bmatrix} \begin{bmatrix} \lambda_{A_1} & 0 & \dots \\ 0 & \lambda_{A_2} & \dots \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{4}$$

and $Q$ is computed in the same fashion.

To eliminate the effect of translation, we center the points by subtracting the centroid of the set from each point, then the centered points is related by the formula $\bar{Q} = R\bar{P}$ where $\bar{Q}, \bar{P}$ are centered data matrices. If the singular value decomposition of $\bar{P}\bar{Q}^T = USV^T$, then $R = VU^T$. The derivation is shown below.

$$\begin{aligned} \bar{P}\bar{Q}^T &= USV^T \\ (U_{\bar{P}} S_{\bar{P}} V_{\bar{P}}^T)(U_{\bar{P}} S_{\bar{P}} V_{\bar{P}}^T)^T R^T &= USV^T \\ U_{\bar{P}} S_{\bar{P}}^2 U_{\bar{P}}^T R^T &= USV^T \\ U_{\bar{P}}^T R^T &= V^T \\ R &= VU^T \end{aligned} \tag{5}$$

The last two equalities is obtained by choosing $U = U_{\bar{P}}, S = S_{\bar{P}}^2$. After getting the result of $R$, the $t$ vector is computed using the following formula.

$$t = -R\mu_P + \mu_Q \tag{6}$$

where $\mu_P, \mu_Q$ are the centroids of $P$ and $Q$. To be robust to noise introduced by the depth sensor and wrong rigid body transformation of the moving object, we have to make an assumption that the majority of the scene is static so that we can run RANSAC to compute the best estimation of $(R, t)$.

Although $(R, t)$ only has six degrees of freedom, we use three pairs of matching key points to compute $R$ and $t$ using the derived formulas 5, 6 because we want to obtain three vectors after subtracting the centroid. If those three points are not colinear, then $(R, t)$ is uniquely determined. Whereas choosing two points would only generate one vector after centering, thus the rotation matrix is ambiguous.

*4) Detecting Changes Using Both Intensity and Depth:* Since the motion of an object causes changes in both intensity and depth, we expected that combining $\Delta I$ and $\Delta\Lambda$ would result to a more accurate detection. Two methods of using both intensity and depth information were implemented, averaging $\Delta = 0.5\Delta I + 0.5\Delta\Lambda$ and $l_2$ norm $\Delta = \sqrt{(\Delta I)^2 + (\Delta\Lambda)^2}$. The result was then compared with a threshold $\epsilon$. If it is greater than the threshold, then it belongs to a moving object.

## IV. Results

### A. Data Generation

For the result of data generation, we have written a class to generate a sequence of human moving images. The results are shown in Figure 5, the person is moving in a straight line at a constant speed. The original HumANav dataset does not support multiple human meshes in one scene, so we added support for multiple human meshes moving in one scene, shown in Figure 6.

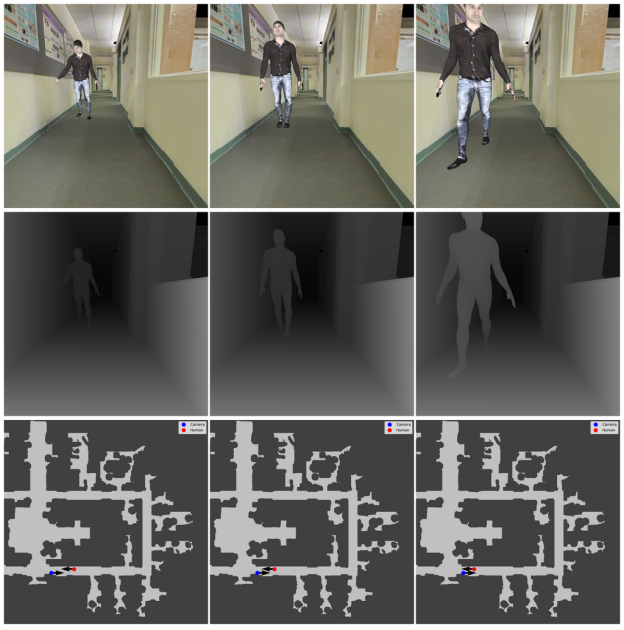

Figure 5. This is a sequence of 10 generated human moving images. The figure shows the 7th, 8th and 9th image of this sequence. The sequence takes total time of 1 second, and the human is moving at $1m/s$. At the top are the RGB images of the sequence, images in the middle are the depth information, images at the bottoms are the map information, including angular and linear velocity of the human, and position of both robot and the human

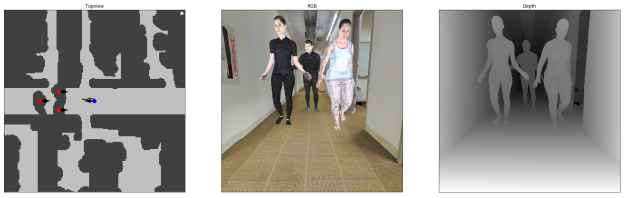

Figure 6. Example of multiple human moving in an environment. In the left in the map information, we added an obstacle area around each human (the darker area around human).

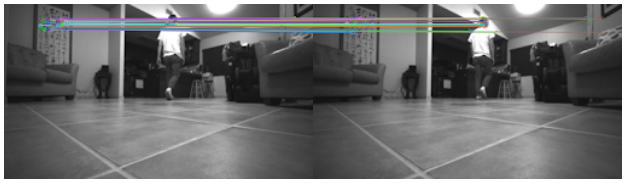

Figure 7. Feature matching of the two input gray-scale images. If we look at the input images $A$ on the left and $B$ on the right, the camera pose has changed by inspecting the tiles on the ground. The bottom intersection of tile lines in $B$ is closer to the image border. The person is walking by checking the change of his pose. In the left image, his left hand is shown; whereas his left hand is covered in the right image.

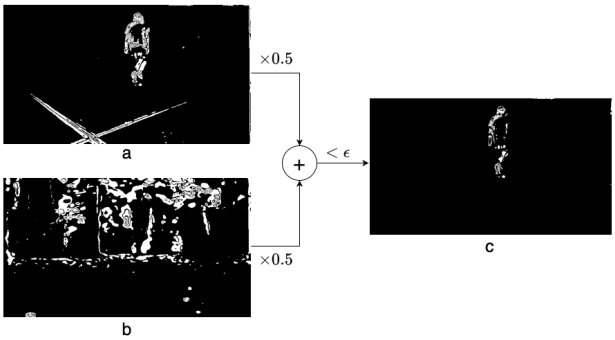


Figure 8. Motion detection result: $(a)$, the changes in intensity $\Delta I$. It also captures the intensity change of the floor, which is caused by the change of camera pose. $(b)$, the changes in depth $\Delta\Lambda$. The depth is sensitive to the setting of the environment. The change of depth captures the border of the walking person and a lot of random stuff from the background. $(c)$, a combination of changes in intensity and depth, produces better and clean detection, where intensity and depth are weighted equally.

### *B. Moving Object Detection*

The input gray-scale images and depth arrays are obtained from an Intel RealSense depth camera D435i, which is mounted on a robot car (designed by the Robot Open Autonomous Racing team) about $10cm$ above the ground. Figure 7 shows the result of feature matching for two input $RGB$ images, in which both the camera pose and the pose of the person have changed. It shows that even after the ratio test, there are still some mismatched features. Intuitively, all lines connecting matched features are not parallel. Thus, it is important to use RANSAC in computing the homography matrix and rigid body transformation.

The results of $\Delta I, \Delta\Lambda, \Delta$ are shown in Figure 8 $(a), (b), (c)$ respectively. $\Delta I$ captures not only the motion of the moving objects but also the motion of the camera. In $(a)$, camera motion caused two lines intersecting on the floor, which is the edges of the tiles in the gray-scale images. $\Delta\Lambda$ captures the borders of the walking person, plus a lot of random things. In a short time interval, the person can only move a little bit, causing little change of distance from his body to the camera. This depth change is below the threshold, so shown to be black in the result. However, the covering and uncovering of the background around his body have changed the depth dramatically, making the border of the moving body clear, which can be seen by comparing $(a)$ and $(b)$. The combination of the changes in intensity and depth $(0.5\Delta I + 0.5\Delta\Lambda) < \epsilon$ provides a much cleaner output and successfully rejects the motion of the camera. In $(c)$, our algorithm is able to capture that only one leg of the person is moving during that time interval. Using $l_2$ norm to combine $\Delta I$ and $\Delta\Lambda$ rather than linear combination provides a very similar result, but it is more computationally expensive.

## V. Conclusion

In this project, we successfully extended the HumANav dataset to allow for a wider range of scenarios that can be used to train our perception module. Rather than having our obstacles take an optimal trajectory, as presented in [6], our dataset allows us to set arbitrary trajectories for obstacles in a simulated environment. For the identification of moving objects, our approach of combining changes in intensity and depth successfully detects the moving objects. The changes in intensity capture both the motion of the object and the camera. The changes in depth capture the border of the moving objects. The linear combination of intensity and depth produces a cleaner detection better than either method individually. We currently plan to feed the results from our motion detection algorithm into a neural network, alongside a corresponding image sequence, a goal position, and the current linear/angular velocity of the robot to get a waypoint prediction.

More work needs to be done in determining a good method for sampling different obstacle trajectories that the vehicle is likely to encounter in the real world. This will be essential in generating a good dataset that our perception module can learn and generalize from it. We also need to experiment with different neural networks in the perception module. At the moment, we are unsure whether a normal CNN as used in [9], a CNN-LSTM, or something different like a temporal CNN will generalize well in this task (and ideally perform well in real-time).

## Acknowledgment

We thank Professor Shankar Sastry and the course staff for assistance with the project idea for the problem. We thank the Robot Open Autonomous Racing team for providing us a robot car and calibrating the RealSense camera. We also thank Nghia Ho for sharing the code on how to compute the rigid body transformation given $3D$ correspondence points.

## REFERENCES


[1] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *CoRR*, vol. abs/1806.10293, 2018. [Online]. Available: http://arxiv.org/abs/1806.10293

[2] H. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end to end," *CoRR*, vol. abs/1809.10124, 2018. [Online]. Available: http://arxiv.org/abs/1809.10124

[3] T. Fan, X. Cheng, J. Pan, P. Long, W. Liu, R. Yang, and D. Manocha, "Getting robots unfrozen and unlost in dense pedestrian crowds," *CoRR*, vol. abs/1810.00352, 2018. [Online]. Available: http://arxiv.org/abs/1810.00352

[4] A. Pokle, R. Martín-Martín, P. Goebel, V. Chow, H. M. Ewald, J. Yang, Z. Wang, A. Sadeghian, D. Sadigh, S. Savarese, and M. Vázquez, "Deep local trajectory replanning and control for robot navigation," *CoRR*, vol. abs/1905.05279, 2019. [Online]. Available: http://arxiv.org/abs/1905.05279

[5] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining Optimal Control and Learning for Visual Navigation in Novel Environments," in *3rd Annual Conference on Robot Learning (CoRL)*, mar 2019. [Online]. Available: http://arxiv.org/abs/1903.02531

[6] V. Tolani, S. Bansal, A. Faust, and C. Tomlin, "Visual navigation among humans with optimal control as a supervisor," *arXiv preprint arXiv:2003.09354*, 2020.

[7] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid, "Learning from synthetic humans," in *CVPR*, 2017.

[8] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese, "Joint 2D-3D-Semantic Data for Indoor Scene Understanding," *ArXiv e-prints*, Feb. 2017.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[10] M. Brown, R. Szeliski, and S. Winder, "Multi-image matching using multi-scale oriented patches," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 510–517 vol. 1.

[11] C. Yu, Z. Liu, X.-J. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei, "Ds-slam: A semantic visual slam towards dynamic environments," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018. [Online]. Available: http://dx.doi.org/10.1109/IROS.2018.8593691