Robotic Room Traversal using Optical Range Finding

(TR2018-991)

Cole Smith
Eric Lin
Dennis Shasha

October 31, 2018

Abstract

Consider the goal of visiting every part of a room that is not blocked by obstacles. Doing so efficiently requires both sensors and planning. Our findings suggest a method of inexpensive optical range finding for robotic room traversal. Our room traversal algorithm relies upon the approximate distance from the robot to the nearest obstacle in 360 degrees. We then choose the path with the furthest approximate distance. Since millimeter-precision is not required for our problem, we have opted to develop our own laser range finding solution, in lieu of using more common, but also expensive solutions like light detection and ranging (LIDAR). Rather, our solution uses a laser that casts a visible dot on the target and a common camera (an iPhone, for example). Based upon where in the camera frame the laser dot is detected, we may calculate an angle between our target and the laser aperture. Using this angle and the known distance between the camera eye and the laser aperture, we may solve all sides of a trigonometric model which provides the distance between the robot and the target.

1 Problem Statement

How can a robot make an efficient traversal of a room with the least amount of passes over the room, and how do we measure distances from obstacles such that the robot traverses all human-accessible areas within the room?

2 Related Work

The complete traversal of robotics through different terrain is not a new problem, and there has been similar work done before. However, these approaches require advanced hardware. We instead propose a cost-efficient manner of room traversal using more common materials.

In a paper from the IEEE 2000 International Conference on Intelligent Robots and Systems, C. Eberst et al. showed that a robot could successfully travel through doorways and avoid obstacles using a multiple-camera array. In addition to optical methods, Eberst et al. also utilizes ultra-sonic sensors and laser scanning for increased navigation reliability [7]. Our implementation differs from these solutions in that we minimized the variety and cost of required sensors.

Another such approach was conducted by Jonathan Klippenstein and Hong Zhang from the University of Alberta, Canada. Klippenstein and Zhang performed research in feature extraction from visual simultaneous localization and mapping solutions (vSLAM) [2]. Similarly, Alpen et al. at the 8th IFAC Symposium on Intelligent Autonomous Vehicles in 2013 explored SLAM features for Unmanned Autonomous Vehicle (UAV) flight for indoor robotic traversal [1], and Sergio García et al. from University of Alcala proposes a solution for aerial vSLAM in a single-camera approach [4]. Our approach uses simpler image transformations and filtering, meaning that our methods are affected less by low computational power, and low camera resolution.

Regardless of the sensory approach for the complete traversal of spaces by robots, Edlinger and Puttkamer at the University of Kaiserslautern propose a solution for an autonomous vehicle to build an internal, two-dimensional map of the traversal space with no prior knowledge about the traversal space itself. In addition to their traversal approach, Edlinger and Puttkamer also leverage optical range finding for navigation [3]. Our approach is only concerned with the traversal geometry of the robot, so the room geometry need not be stored for our algorithm.

3 Materials

3.1 Hardware

- ARM, x86-64 OS for Go programs
- iPhone 6s Plus
- Generic iPhone Suction Car Mount
- Stepper Motor (Any step count, 12v)
- iRobot Roomba model 600

3.2 Software

- Room Traversal Algorithm
- EasyDriver board for Stepper Motor
- GPIO Driver for Roomba

3.3 Cost Analysis

The system is designed to be as cost effective as possible. Our camera solution total cost was less than $90. The cost, without camera or robotic platform, can be broken down as:

- Raspberry Pi Model 3B: $35
- Generic Stepper Motor: $12
- EasyDriver Stepper Motor Driver: $14
- Generic iPhone Suction Car Mount: $8
- Plastic housing: $11
- **Total Approximate Cost: $80**

An iPhone is not required to use the camera interface. Any camera can be used so long as it can export PNG files to our system. For example, the Raspberry Pi Camera retails for $26 as of 2018. Comparable LIDAR solutions can cost > $300[6].

4 Hardware Implementation

4.1 Robotic Testing Platform

For our robotic platform, we are using an iRobot Roomba Model 600. Direct control is assumed over the Roomba using the provided serial port at the top of the unit. Our system implements a module that sends debug commands over the Raspberry Pi GPIO pins as serial output.

The Roomba is set to manual-drive mode using a specific serial command, and then subsequent move commands are sent to the Roomba when required. Since the Roomba is always set to move at a constant speed, location can be measured by counting the encoder values of the Roomba's wheels and comparing the value to the total time that the wheels are turning.

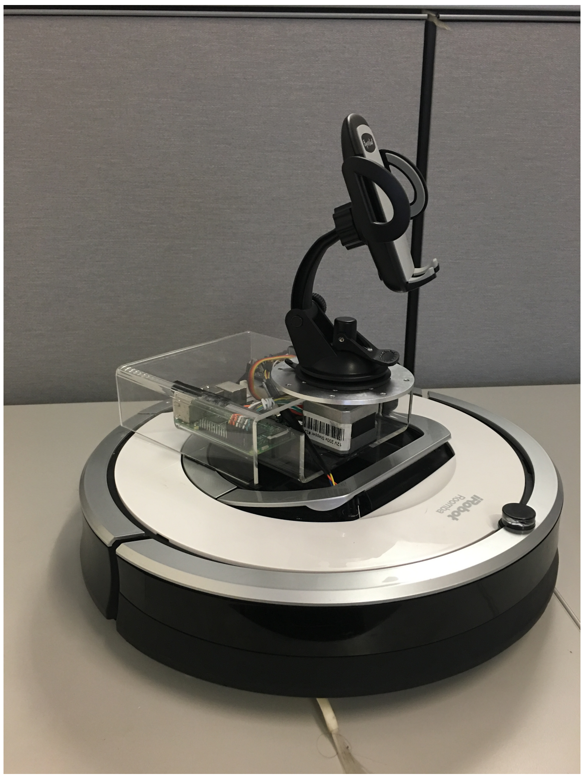

Figure 1: Robotic platform using Roomba 600 model

4.2 Camera Testing Platform

The flow of image processing is as follows:

1. Image is streamed from iPhone to program as PNG, using HTTP server and client programs. (CameraStreamer iPhone Application)
2. PNG is converted to 2D pixel array in HSV color space.
3. Array is passed through luminosity thresholding filter.
4. Array is passed through color thresholding filter.
5. Blob detector is run on array, and the centroid of the blobs are detected.
6. Ovals are rejected from the system, leaving only the laser dot centroid.
7. The offset from the vertical center of the camera plane and laser dot is modeled as an angle, and used to calculate distance from obstacle to robot.

5 Laser Dot Detection

5.1 Data Flow and Filtering

PNG files are streamed from the CameraStreamer iPhone application to our program and converted into a 2D matrix in HSV (Hue, Saturation, Value) color space. Pixel values are stored as a struct, and each Hue, Saturation, and Value variable are normalized to be in the range [0,1]. This matrix will undergo a series of filtering steps to convert it into a binary image mask. Three filters are used to detect a laser dot within an image: Luminosity, Color, and Oval Rejection. Each filter defines a set of thresholds and target values for which to convert the HSV matrix into a boolean matrix.

5.2 Luminosity Filtering

The first pass through the HSV matrix filters away pixels of undesired luminosity. For our purposes, we select only the brightest pixels within the image, since those are likely to represent the laser dot in the image frame. The conversion function creates a mask where "true" values are defined for pixels above or equal to the threshold value, and "false" for pixels below the threshold value.

```
func (image ImageMatrix) ConvertToMonoImageMatrixFromValue(valueThreshold float64)
*MonoImageMatrix
```

The conversion function defines a method on the `ImageMatrix` struct and takes its Value threshold as an argument, `valueThreshold`. This float defines the minimum cutoff for the Value (of HSV) of a pixel, in normalized range of [0,1]. The function then returns a pointer to a `MonoImageMatrix` struct. This struct masks pixels of insufficient luminosity.

5.3 Color Filtering

A second pass is made over the HSV matrix to filter Hue values within the HSV color space. A target hue and a hue threshold are defined in which pixels are masked if the absolute value of the difference between the hue and the target hue exceeds the hue threshold.

```
func (image ImageMatrix) ConvertToMonoImageMatrixFromHue(hueTarget, hueThreshold float64)
*MonoImageMatrix
```

The function defines another method on `ImageMatrix`. A pointer to a `MonoImageMatrix` is returned, as it was in the previous luminosity filtering, with the pixels that deviate further than `hueThreshold` from `hueTarget` masked, and represented visually as black.

The two masks, luminosity and color, are then combined into one `MonoImageMatrix`. Rendered, a masked image of a laser dot will appear like the following:

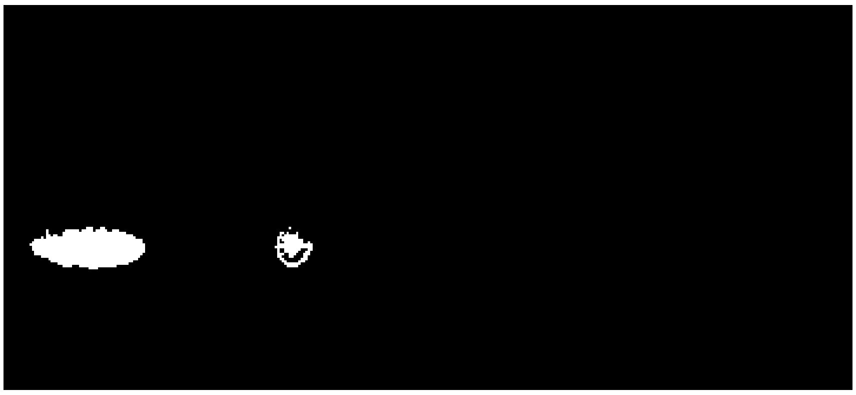

Figure 2: A filtered image showing a green laser dot and reflection

5.4 Blob Detection

The masked matrix is then traversed using a 4-connected blob detection algorithm. Small artifacts can be rejected by defining a `MIN_BLOB_SIZE` constant in number of pixels. The blobs are then returned as an array of pixel coordinate groups (X,Y) which are connected and "true" within the boolean image matrix. In figure 2, two groups of pixels will be returned.

5.5 Oval Rejection

The filtering passes will successfully mask out all light that does not conform to the luminosity and color profile defined as thresholds to the filtering methods. This will leave a boolean mask of the laser dot and any reflections of the laser dot in the image frame. Since these reflections will often appear less circular than the laser dot itself, we may reject the blobs that do not conform to a defined circular ratio.

```
// Given a series of connected coords, take the difference of
// min and max values for X and Y. The differences for X and Y
// are made a ratio as:
//                 [ abs(minX) - abs(maxX) ] / [ abs(minY) - abs(maxY) ]
// or
//                 [ abs(minY) - abs(maxY) ] / [ abs(minX) - abs(maxX) ]
// A ratio of 1.0 denotes a perfectly square bounding rectangle,
// (a circle blob). Anything less, denotes the oval ratio
func getCircleRatio(blob []*coord) float64
```

In figure 2, the leftmost oval will be rejected from consideration as a laser dot, leaving only the rightmost blob. The centroid of this pixel coordinates group is then calculated. The pixel distance of the centroid to the vertical center of the camera plane is then used to determine the physical distance between the laser dot and the laser diode.

6 Range Finding Model

6.1 Calibration

The range finding program must first be calibrated before it may be used. Two methods may be used: (1) One calibration step is used, with the camera rotated with every distance reading during normal operation, to determine the angle needed to match the laser dot to the vertical center of the camera plane. (2) Multiple calibrations steps are used, with the camera not rotating while taking distance readings during normal operation, to determine the rate at which the laser dot moves away from the vertical center of the camera plane.

Using method (1), the user will place the robot such that the laser diode is 1 unit of distance (1 meter, 1 foot, etc.) away from the laser dot projected on a clean surface. The camera will then rotate using the stepper motor until the laser dot converges to the vertical center of the camera plane. The angle of rotation found in this calibration step defines a triangle for which the side opposite to the hypotenuse is one unit of measure. Subsequent measurements during normal operation will be defined as a continuous proportion of this unit measure.

Using method (2), the user will originally place the robot as they would in method (1). More than one calibration steps are used, and the user will then place the robot at 2, 3, and 4 units of measure away from the laser dot. At each calibration step, the distance of the laser dot to the vertical center of the camera plane is recorded, and the rate of change in pixel distance is determined as the robot moves further away from the laser dot. This method of calibration has the limitation that the physical distance reading during normal operation is only as accurate as the approximated rate of change in pixel distance determined by this calibration method. For distances greater than the N units of measure conducted during calibration, the distance measured will be extrapolated from this approximated rate of change. For this reason, our implementation supports method (1) only.

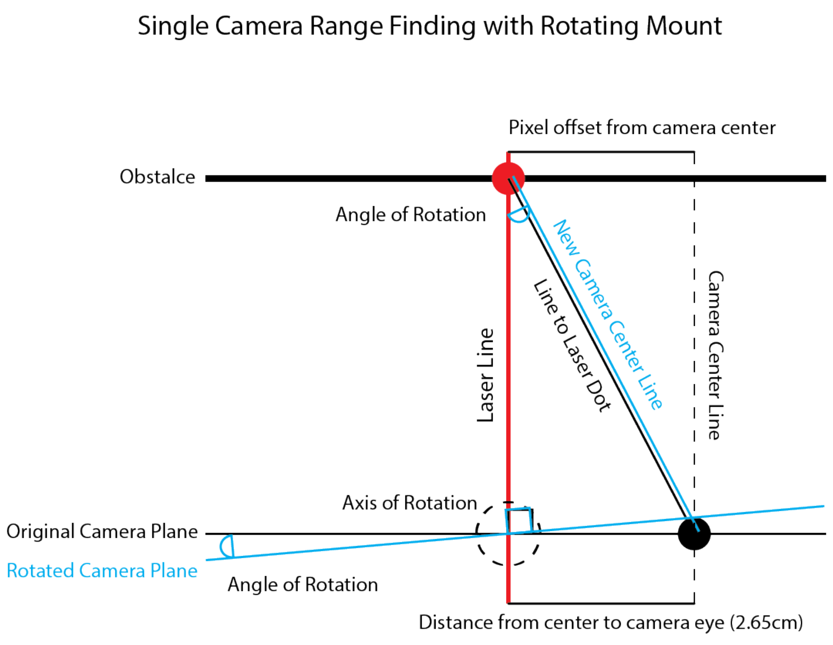

Figure 3: Triangular Distance Model

6.2 Range Calculation

For each distance reading, the camera is rotated until the laser offset from the vertical center of the camera plane is minimized, or ideally zero. The angle of rotation required is recorded, and given to a simple function to solve the triangular model in Figure 3 using the Sine Law. The camera rotation is performed using the stepper motor and iPhone mount seen in Figure 1

```
// Returns the distance from the laser diode to the target based upon the
// provided angle, at which the pixel offset was corrected by rotating the camera
// such that the laser dot was in the center plane of the camera.
func GetLaserDistance(angle float64, triangleBase float64) float64 {
        sineLawBase := (triangleBase / math.Sin(angle))
        sineOfAngleC := math.Sin(90 - angle)

        return sineLawBase * sineOfAngleC
}
```

Limitations The constraints of our range finding model have theoretical limits based upon camera resolution and stepper motor resolution (how many discrete "steps" are available in 360 continuous degrees). As camera resolution increases, the model can gauge distance further as the increased pixel count allows for greater room between the laser dot and vertical camera plane, such that the vanishing point (where the laser dot and vertical camera place will naturally converge as distance increases) will be further from the laser diode. While greater resolution will increase computation time polynomially, more possible steps within the stepper motor allow for a more accurate angle when rotating the vertical camera place to the laser dot.

7 Room Traversal Method

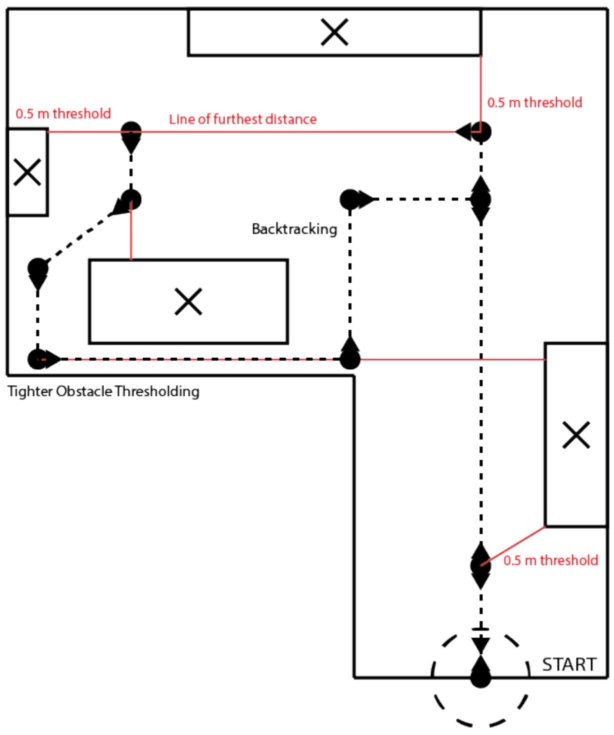

Figure 4: Room Traversal Model with Range Finding

Based on the above image, the user will pick a starting point in a given room. The robotic vehicle will then scan the surrounding area in 60 degree increments for the direction it can travel the furthest, giving 6 possible directions of travel. Once a direction has been determined, it will start its navigation towards that direction, keeping a predetermined amount of threshold between the vehicle and other potential obstacles.

The vehicle continues down the direction until the threshold eventually stops the vehicle from traveling in that direction, and then it scans the room again for the furthest direction to travel without backtracking.

The vehicle will eventually reach a point where it cannot move forward without backtracking, and once that point is reached, it will first decrease the obstacle threshold and determine whether it allows the vehicle to move in additional spaces it has not been to before. An algorithm can be defined as follows:

1. START at doorway or accessible entrance
2. Take 6 distance readings in 360 degrees and begin traversing the path of maximum distance
3. Stop when distance to obstacle in the forward travel direction is less than threshold distance
4. Take 6 distance readings in 360 degrees
5. If all distance readings are below threshold, temporarily lower threshold to maximum distance of previous reading
6. If threshold has been lowered to less than the width of the robot (the robot can no longer traverse into a space), backtrack out of space by following previous line of travel, goto 4.

7. Else if forward movement crossed a line of previous traversal (cycle detected), stop, find path to starting position using previously traversed paths, follow path, END.

8. Else, goto 2

Once all available space has been traversed, it will then return to the starting position through the nearest path it can find to return.

8 Results

We compared our approach to a naive-bounce approach, in which the robot will make 90 degree turns when bumping into an obstacle. The traversal concludes when the robot reaches its starting position.

We found our approach to offer improved traversal since it prevents the robot from entering infinite bounce-loops, or ending its traversal early, as seen in the figure below.

The below results compare our algorithm to the naive bounce approach. The grey blocks represent obstacles. Yellow lines denote the robots traversal path, which begins in the lower-right corner. For our algorithm's approach, the initial obstacle threshold was set to 20 units. The

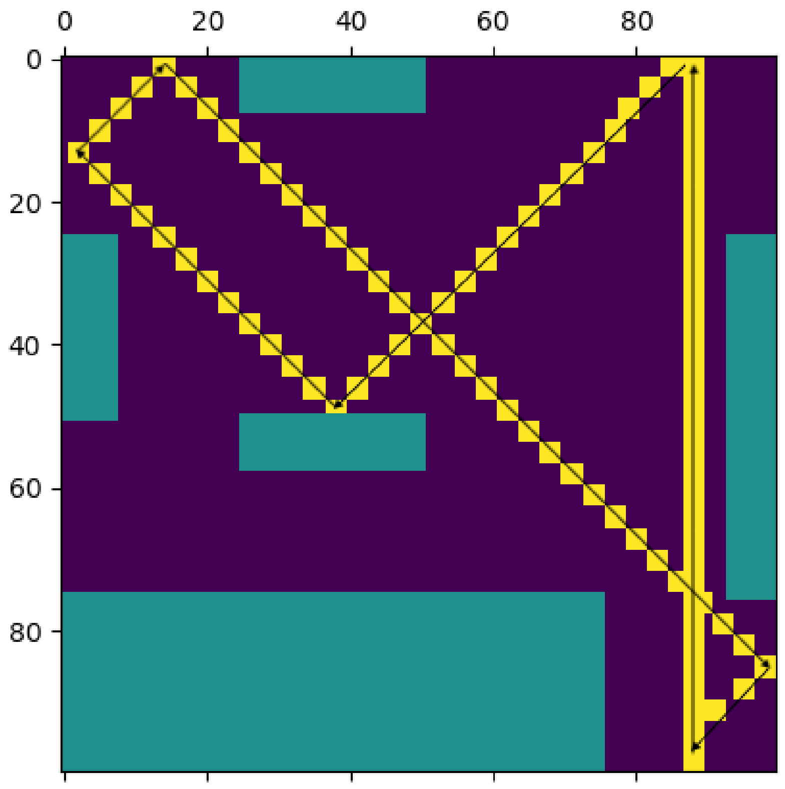

Figure 5: Robot Traversal using Naive Bounce (90 Degrees)

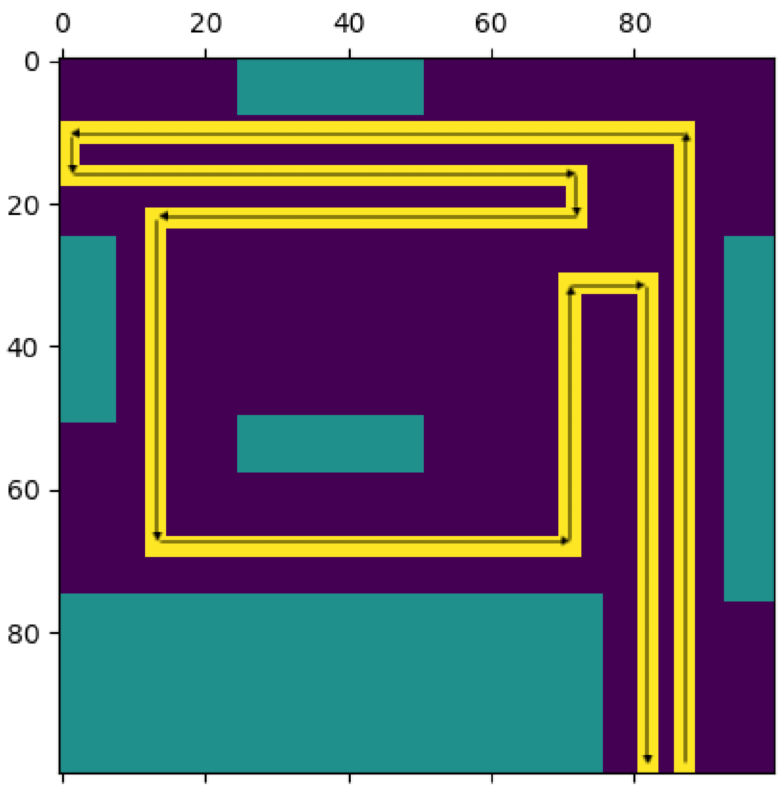

Figure 6: Robot Traversal using our algorithm (threshold: 20 units)

9 Limitation: No Outdoor Robotic Traversal

Our current robotic platform will not perform well in rough terrain outside of an office or home setting. Our system assumes flat ground with distance determined in 2 dimensions around the robot. As the robot encounters rougher terrain, a pitch in the Y dimension will be introduced. In order to properly handle outdoor situations, our model will have to be adjusted to incorporate distance calculation in 3 dimensions. This could be achieved by measuring the pixel distance between the laser dot and the horizontal center of the camera plane.

10 Parallel Camera Support

Our language choice (Go) naturally allows for more than one camera to operate in parallel. As a further step, more cameras and mounts can be added behind, or to the sides of the robot to decrease the need for the robot itself to rotate to gauge distance in 360 degrees. Currently, the laser itself is fixed, so the robot must rotate its entire assembly to point the laser at a different obstacle. Additionally, distance can be determined in 3 dimensions by adding cameras and lasers pointing upwards or at a pitched angle.

11 Conclusion

The current mono-camera SLAM has been tested and shown to be successful in a flat indoor areas. Our system provides an autonomous and cost-effective solution to room traversal in stable environments. For further considerations, we would like to improve the ease of threshold tuning for the machine vision pipeline model, and expand the CameraStreamer iOS application to provide control over the entire system. In doing so, we would allow our system the first steps into less stable environments such as outdoor scenarios with high brightness. In extremely bright environments, our model will support the use of IR lasers and cameras. Additionally, the pitch of the camera in the Y dimension would need to be considered for non-flat terrain. Given these issues are addressed, this research provides future expansion into areas such as unmanned aerial vehicle navigation, since our long term considerations include abstracting our model to a more general method of environment traversal.

We provide a basic SLAM scaffold for any robotic vehicle using a single camera setup on our repository home page:

`https://github.com/NYU-Efficient-Room-Traversal`

12 Acknowledgements

This research was supported in part by the funding of NYU College of Arts and Sciences Dean's Undergraduate Research Fund and NYU WIRELESS.

References

[1] Mirco Alpen, Klaus Frick, and Joachim Horn. An autonomous indoor uav with a real-time on-board orthogonal slam. *IFAC Proceedings Volumes*, 46(10):268 – 273, 2013. 8th IFAC Symposium on Intelligent Autonomous Vehicles.

[2] C. Eberst, M. Andersson, and H. I. Christensen. Vision-based door-traversal for autonomous mobile robots. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 1, pages 620–625 vol.1, 2000.

[3] Thomas Edlinger and Ewald von Puttkamer. Exploration of an indoor-environment by an autonomous mobile robot. Technical report, Department of Computer Science, University of Kaiserslautern, Erwin-Schrödinger-Straße, P.O.Box 3049, D-67653 Kaiserslautern, Germany, September 1994.

[4] S. García, M. E. López, R. Barea, L. M. Bergasa, A. Gómez, and E. J. Molinos. Indoor slam for micro aerial vehicles control using monocular camera and sensor fusion. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 205–210, May 2016.

[5] Arturo Gil, Oscar Martinez Mozos, Monica Ballesta, and Oscar Reinoso. A comparative evaluation of interest point detectors and local descriptors for visual slam. *Machine Vision and Applications*, 21(6):905–920, Oct 2010.

[6] Tony Huang. Rplidar a1 - slamtec - leading service robot localization and navigation solution provider.

[7] J. Klippenstein. quantitative evaluation of feature extractors for visual slam. *Proc. the Fourth Canadian Conf. Computer and Robot Vision, Washington DC, USA, 2007*, pages 157–164, 2007.

[8] H. Lategahn, A. Geiger, and B. Kitt. Visual slam for autonomous ground vehicles. In *2011 IEEE International Conference on Robotics and Automation*, pages 1732–1737, May 2011.

[9] B. Steder, G. Grisetti, C. Stachniss, and W. Burgard. Visual slam for flying vehicles. *IEEE Transactions on Robotics*, 24(5):1088–1093, Oct 2008.

[10] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, Jun 2017.