Abstract

Sai Nihar Tadichetty, Joao Fonseca Reis, Luka Eerens, Keerthana P G

Finding your way using only raw data from the

Animals move around the world in complex unstructured environments, seamlessly transiting between places where they have been before, and thus a prior is available, and other places they have never been to before. They operate for hours on end and simplify the overwhelming data gathered down to a few essential data inputs that allow them to operate effectively without being overflown with excessive information (in particular visual stimulation). In this paper, we present an extremely simple algorithm that allows any entity to move from an initial point to a target away point in an environment cluttered with multiple types of objects using only raw information from sensors as input without even the need to build a map. With the additional bonus that it is able to work real time, the robot will never get trapped in local minima, will typically chose the shortest path, or close to it, and will always find its own way provided that a solution is possible given the robot constraints (e.g. size to fit through spaces in between the obstacles).

***Index Terms*— no prior knowledge of environment, raw sensor data, avoiding obstacles, circling obstacles, not trapped in local minima, real time operation**

I. INTRODUCTION

Our main objective is to develop a simple algorithm whereby a robot can successfully navigate from point A to point B by relying solely on processing real time sensor information without any prior knowledge of the environment where it operates. A resilient algorithm whereby human sets the robot away point and the robot simply "figures out" how to get there without any human in the loop micro-managing has endless applications in drones, whether for missions of surveillance, mapping, delivery of cargo and people and many other applications we cannot phantom today.

Hybrid systems were humans and machines are blended together in a loop have become a reality in recent years, with systems designed around the concept that humans prefer to be comfortably in control, without having to supervise every minute detail of operation [1]. This is akin to the way the human mind operates [2] [3] whereby the higher levels of the mind plan a path and most of the implementation and local control is delegated (down to the proper level in the hierarchy of the human brain) so that human abstract thinking is not overflown with needless detail.

A. Past Work in obstacle avoidance algorithms

The usage of vector fields to guide a robot was pioneered by Ronald C. Arkin, among others, in the 90's and is normally referred to as "behavior-based-robotics". The proponents of this approach foster dynamic planning in an unstructured environment as the basis for the definition of simple policies a robot can follow as the environment changes around it.

In [4] and [5], this approach was materialized using a multilayered hierarchical architecture based on potential vector gradients that would drive a robot to a target, guide its avoidance of an obstacle, and perform successive tasks as distant gates were actioned (e.g. carry bucket after reaching it).

B. What is different about the proposed algorithm

The main issue with using a gradient as a guidance field is for obstacles, is that the robot can get trapped in local minima when operating in environments cluttered with obstacles. This happens as the field of several neighboring obstacles combines to generate points where the guidance field divergence is negative (where field lines "die").

In addition, the propose approaches in [4] and [5] typically involve integrating across multiple states and using algorithms akin to A star to select the path with the biggest payoff. This approach becomes particularly expensive in particular when on or more of the following conditions is verified: highly cluttered environments (where previously occluded information may imply planning dramatically different paths once an obstacle is overcome), multiple sensors fused together with dense data readings, robots that travel at high speeds.

By relying solely on local information with map or forward multicycle integration, the presented algorithm will work mostly in constant time, thus making it more resilient to dramatic changes in the environment where the robot operates.

C. Main intuition behind the algorithm design

The algorithm proposed on this paper is based on computing a curl of a vector field, which does not generate local minima, and combining that with a gradient field point to the target where the only local minima will be located. In addition, to ensure that the robot does not "orbit" the objects longer than it should, the contribution of that obstacle to the guidance field is" disconnected" whenever it is deemed irrelevant for the task at hand: avoiding obstacles while going from point A to B.

Finally, the algorithm works in two different modes: sparse and dense environment, seamlessly switching between the two modes. When in dense environment, the field rotation is kept constant for all sources (e.g. they all rotate counter-clock or clockwise). When in sparse environments, the fields rotate in a way that pushes the robot to pass in between the obstacles.

As a proof of concept, we applied the algorithm in simulation to SR-10 toy tank robot. This robot has enough authority of control to rotate in place and thus allows for tremendous freedom of movement and authority of control,

that would not be available for example in a quadcopter. As a result, we have used a Gaussian profile for the generation of the "obstacle fields" given that it is a well-known function with fast decay, thus ideal for creating localized fields that don't spread too far from the obstacle.

For the application of this algorithm to a flying quadcopter, or any other vehicle with decreased authority of control, the algorithm would have to be adapted so that the obstacle field would scale with the available authority of control for the current dynamics. This would ensure that the guidance fields would not issue a command the quadcopter was not able to implement (for example, making a ninety degrees curve in mid-flight).

II. Detailed description of the proposed algorithm

The "guidance field" (G) is defined as:

$$\vec{G} = \vec{T} + \vec{H} + \vec{S}$$

where three different fields are linearly combined together:

- "Target Field" (T): defined by the way point (can be changed at any time by the user)
- "Hard obstacles field" (H): coming from the LIDAR hits, corresponds to obstacles that will not allow the robot progression (e.g. walls)
- "Soft obstacles" (S): coming from the camera hits, corresponds to obstacles that cannot be captured by the LIDAR, but should also be avoided (e.g. sand or grass, ink markings on tarmac, ….)

Both the obstacle guidance fields are dynamically computed at every iteration and orthogonal to the plane in which the robot operates. In both cases, their value will depend on the obstacle distance from the robot and its relative bearing position.

A. Target Guidance Field

The Target Field is defined as the stationary gradient field with local minima at the target point. This field was defined as independent from the target distance, with constant norm, in order to prevent conflict with the two other fields. The scaling constant should be small compared to the other two fields to give priority to the obstacle fields whenever needed.

$$\vec{T} = \vec{\nabla}\Phi = T_0 \frac{(x_T - x_0, y_T - y_0, 0)}{\sqrt{(x_T - x_0)^2 + (y_T - y_0)^2}}$$

where the target state is denoted by (x_T, y_T) and the current state coordinates by (x_0, y_0). The Obstacle Fields are both of the same exact form and differ only on the scaling constant to be used so that different level of responses can be enacted for different types of obstacles. These last two fields are created from sensors data and treated separately so that different weights can be given to the two types of obstacles: "hard" and "soft".

B. Hard obstacles guidance field:

Each LIDAR hit will generate a different "hard obstacle field" that will be linearly combined across all N hits recorded at a particular instant:

$$\vec{H} = \sum_{i=0}^{i=N} \vec{H_i} = \vec{\nabla} x \vec{A} = \vec{\nabla} x \left(0,0, H_O e^{\frac{-(x_i - x_0)^2 + (y_i - y_0)^2}{\sigma^2}} \right) =$$

$$\sum_{i=0}^{i=N} (y_i - y_0, -x_i + x_0, 0) H_O D_i e^{\frac{-(x_i - x_0)^2 + (y_i - y_0)^2}{\sigma^2}}$$

where, H_0 and σ will be constant across all hard obstacles, describing respectively the maximum intensity of the hard obstacles field and its decay rate with distance, Di is a variable that can take one of three values: -1, 0 or 1 depending on the situation at hand.

C. Soft obstacles guidance field:

Each camera hit will generate a "soft obstacle field" that will equally be linearly combined across all M hits recorded at a particular instant:

$$\vec{S} = \sum_{j=0}^{j=M} \vec{H_j} = \vec{\nabla} x \vec{B} = \vec{\nabla} x \left(0,0, S_O e^{\frac{-(x_j - x_0)^2 + (y_j - y_0)^2}{\sigma^2}} \right) =$$

$$\sum_{j=0}^{j=M} \left(y_j - y_0, -x_j + x_0, 0 \right) S_O D_j e^{\frac{-(x_i - x_0)^2 + (y_i - y_0)^2}{\sigma^2}}$$

This feature was tested in simulation to prove that the algorithm would work seamlessly with different types of fields, but not rolled out to the final robot setup.

D. Computing the direction of rotation (Dj)

Depending on the mode selected, the algorithm will either use a common direction for all fields sources (dense mode) or differentiate the factor for each of the obstacles (sparse mode). While operating in the dense mode, the rotation direction will remain constant for all sources and equal to the previous step.

While operating in the sparse mode, for both the Hard and Soft fields, the "direction factor correction" (Di) is defined for each obstacle as:

$$\vec{D_j} = max \left[0, \vec{v} \cdot \left(r_j - r_0 \right) \right] =$$

$$max \left[\left[0, \left(v_{x0}, v_{y0}, 0 \right) \cdot \left(x_j - x_0, y_j - y_0, 0 \right) \right] \right] =$$

$$max\left[0, v_{x0}\left(x_j - x_{0T}\right) + v_{y0}\left(y_j - y_0\right)\right]$$

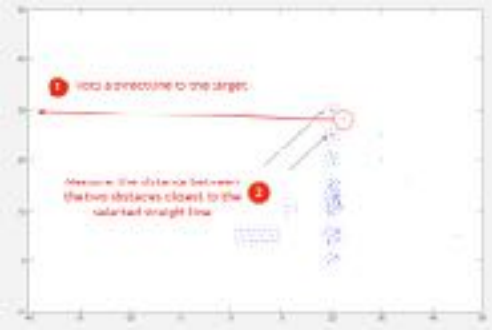
where the dot product of the robot speed (V) and the vector point from the robot to the obstacle is computed to ensure the robot circles the obstacle taking the most appropriate option (for example opting for circling the object through the left of right sides).

The D factor by default switches off the obstacles that are behind the robot, and thus irrelevant for its mission. The definition of "behind" is any object in the opposite direction of the robot movement (defined by its vector speed) regardless of the robot orientation at the moment tracked in the state vector.

E. Switching between the sparse and dense modes

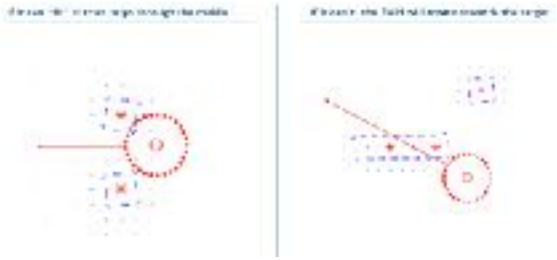
The algorithm starts by drawing a direct line between the current position and the target position. Along that line, the algorithm measures the distance between the first obstacle above and the first obstacle below that line. Only the closest obstacles are relevant:

Figure 1: Detecting relevant obstacles for mode selection



If the obstacles are separated by enough distance, the algorithm will operate in sparse mode. Otherwise it will operate in dense mode. To avoid switching back and forth between the two modes, a hysteresis cycle if forced by defining a bigger distance to switch from dense to sparse (we defined a factor of two) than from sparse to dense.

Figure 2: Behavior induced across dense and sparse modes.



F. Input data to the algorithm

The inputs from the LIDAR and the camera are transformed into a point cloud of hits that serve as inputs to the algorithm. By operating continuously with only the latest raw data available from the sensors, the robot is able to actively react to changes in the current environment, even avoiding dynamic obstacles. In the present form the robot is not able project the movement of obstacles but can seamlessly react to their new positions in the world frame.

The hits from both the LIDAR and the camera are transformed into world coordinates in meters so that it can be fed into the guidance algorithm. This data is combined with the data flowing from the IMU and the localization development boards to access the current state and decide the next step.

G. Simplified motor dynamics used in this paper

In [6] a detailed deduction is presented for the interplay between the electric and mechanical load on an electric motor. Assuming the mechanical impedance is much larger than the mechanical impedance, the angular speed of a linear motor is:

$$\dot{\theta}(t) = \dot{\theta}_0 e^{-\gamma * t} + \dot{\theta}_{st} * \left(1 - e^{-\gamma * t}\right)$$

The decay constant (γ is intrinsic to the system, while the steady speed will be determined by the load on the motor. For the purpose of this paper, we will ignore any transients and issue the motor controls as if the robot was always in steady state.

This approach was approach was successful implemented in [7] and [8] where a steady state derived algorithm was used to in the TAEM section of the Space Shuttle's reentry flight simulation. This can be applied whenever convergence to the steady state is fast relatively to the trajectory transverse by the robot, the controls inputs issued by the algorithm are smooth and motor controls are not buffered any circumstance (in case they cannot be executed old control inputs should be immediately replaced with the latest control).

H. Motor inputs to align robot with guidance field

The SR 10 is a differential drive robot that can move forward or rotate in place. Under a simplified kinematics approach described in [9] the motion model can be simplified as a rotation about an ICR (instantons center of rotation).

Under this framework, when moving in a straight line, this point moves to infinity. However, the robot speed is well defined at all moments and with some algebraic manipulation it can be shown that under steady state:

$$\begin{cases} V_x = \frac{1}{2}\left(V_L + V_R\right)cos\left(\theta_0 + \omega_{st}t\right) \\ V_y = \frac{1}{2}\left(V_L + V_R\right)sin\left(\theta_0 + \omega_{st}t\right) \\ \omega_{st} = \frac{1}{L}\left(V_R - V_L\right) \end{cases}$$

where V_x and V_y are the robot speed components, ω_{st} the steady state angular speed, V_R and V_L are respectively the speeds of the right and left wheels of the robot, L the distance between two wheels sharing the same axes.

Whenever the guidance field is computed, the target alignment angle is derived from the guidance field at the robot's current location using:

$$\theta_{target} = atan2\left(G_y, G_x\right)$$

The motor control than issues a command to the motors that will deliver the desired angle in steady state and readjust its control input as the robot moves towards the intended location. The misalignment angle is computed by the difference between the target alignment and the current alignment:

$$\delta\theta = \theta_{target} - \theta_0$$

And the motor controls inputs are then given as per the following formulas dependent on current misalignment angle:

if $\theta_{Target} >= -\pi$ and $\theta_{Target} < -\pi/2$
  Motor_Left_Multiplier= 2 + (θ_{Target} +π/4)/(π/4);
  Motor_Right_Multiplier= - 1;
end

if $\theta_{Target} >= -\pi/2$ and $\theta_{Target} < 0$
  Motor_Left_Multiplier= +1;
  Motor_Right_Multiplier= 1 + θ_{Target} /(π/4);
end

if $\theta_{Target} >= 0$ and $\theta_{Target} < \pi / 2$
  Motor_Left_Multiplier= 1 - θ_{Target} /(π/4);
  Motor_Right_Multiplier= 1;
end

if $\theta_{Target} >= \pi/2$ and $\theta_{Target} <= \pi$
  Motor_Left_Multiplier= - 1;
  Motor_Right_Multiplier= 2 - (θ_{Target} -π/4)/(π/4);
end

These commands are issued as target ratios against the maximum speed that are then transformed according to the motors torque curve that was previously calibrated in the lab.

Given that the control is the acting solely on the robot orientation, the algorithm as an extra degree of freedom which is the actual speed of the robot. In this project, we set this target cruise speed at the beginning of every run.

In this robot, with full authority of control including the ability to make turns in place, the algorithm is invariant against the target speed selected at the beginning of the run (as this target speed will only be achieved while travelling in a straight line).

III. Systems Architecture Under Implementation

The systems architecture developed for this system can be summarized from a high-level point of view by the following diagram, that includes the primary owner of each module:

Figure 3A: Functional architecture

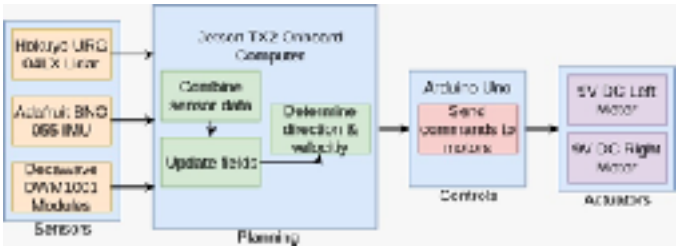


Figure 3B: Cyberphysical architecture

Most software is written from scratch, reusing only standard ROS messaging modules and sensor data retrieval packages defined by the manufacturer. The algorithms are also written by the team, specifically for this project, including the motors calibration controls.

The full list of packages selected for the project is:

1. OpenCV for Python
2. ROS urg_node
3. PyTorch
4. Inferno
5. DWM1001 packages
6. BOSCH BNO055 IMU libraries

A. Robot body frame sub-system

This project was mounted on top of the body frame of a scaled SR 10 tank, that we have deeply customized for the purposes of this project. We used the original frame of the or SR 10 system as a starting point and rebuilt its entire electronics for the purpose of this project.

B. Central sub-system

The central sub-system is the brain of the system, in charge of parsing all the data from the sensors, transforming it into world coordinates to compute the guidance fields, computing the guidance fields and transforming that into a motor input to drive the robot to the next state in tandem with the target state defined and the existing obstacles. This central sub-system is powered by a Jetson TX2.

- Inputs: target state, LIDAR, camera, IMU (current state estimation)
- Outputs: Guidance field, Motor commands

C. LIDAR integration

The LIDAR is the sensor in charge of finding all the hard surfaces the robot needs to avoid. It is powered by a Hokuyo Urg-04lx scans the 2D plane the robot operates in.

- Inputs: reflections from infrared lasers fired by the LIDAR
- Outputs: distance and bearing hits of hard obstacles.

D. Vision system

The vision sub-system scans the textures of the surrounding environment to detect soft obstacles that can not show up in laser range finders scans (LIDARs), such as grass, sand or markings on the tarmac. This sub-system is powered by the Microsoft HD cam designed for outdoor operation.

- Inputs: visible light from the environment
- Outputs: digital images that will be transformed into point clouds of soft obstacles by the central sub-system.

Note: The vision system was descoped in our robot implementation as it was a stretch goal using a neural network.

E. State estimation system

This sub-system is composed by two high precision IMUs, one measuring the position (DWM1001 Development boards for localization) and the other the orientation (Adafruit BNO055). We also have access to the wheel encoder acting as secondary system (that is not as reliable given that we will be operating indoors where wheel slip makes it less reliable and is described in the motors section). The IMU sub-system is powered by the DWM1001 Development boards for localization.

- Inputs: inertial readings
- Outputs: current location, linear speed, angular orientation, current angular speed

F. Motor control and calibration

The motors sub-system is made of three components: the Arduino UNO coupled to a drive board to control the motors, a hall effect to monitor the motor rotational speed and the DC motors. The system was designed to operate real time without any buffer so that commands that cannot be executed in time are simply ignored for the new commands to fed into the motors. This is of particular importance for a real time operation algorithm such as the one being implemented in this project.

Arduino UNO + Motor Drive board

- Inputs: motor commands from the central sub-system
- Outputs: motor commands

9v DC motors with attached encoders

- Inputs: motor commands from the Arduino UNO
- Outputs: torque to move the robot

Hall effect wheel encoders

- Inputs: wheel turning readings
- Outputs: wheel turning readings

G. Results obtained: robot setup

We finalized all hardware and software deployment in our SR10 robot. We opted for ROS based environment so that we could seamless integrated multiple sensors operating at different frequency rates.

We have also tested in simulation the first version of the algorithm (developed in Matlab) and ported it to Phyton to be deployed in the ROS environment that was setup in the Jetson TX2. The initial prototype was done in Matlab so that we could leverage all the graphical abilities during the debugging and testing phases. Phyton was selected as deployment language due to its fast performance.

The IMUs were selected so that we could operate indoors and outdoors with high precision. The development boards are connected by Bluetooth. We opted for complementing these with a traditional IMU so that we could leverage, at a low cost, the best state estimations both systems could provide.

The motors calibration was done with and without load. This proved particularly important given that a low energy dead-band was detected, different responses to the same input power were detected across the two motors, and their transients were confirmed to be fast enough to be ignored for the purpose of this project (a robot operating on low speeds <3 m/s).

The Hokuyo LIDAR was selected due to its affordability and the fact that it only generates a 2D scan, thus a small point cloud that will allow us to focus the computational power of the Jetson TX2 on the controlling the overall system (and not only on parsing 3D point clouds as would be the case if we had for example selected a Velodyne).

The HD camera was selected by its ability to operate in an outdoors environment across multiple lightning conditions, the plug and play feature and its low weight. This was one of our stretch goals and was made available for future work although we did not connect it to the algorithm and the neural network we started working on.

H. Results obtained: Algorithm in simulation

The first algorithm runs were made in Matlab using a simplified Euler numerical integration with the following parameters:

- Target state = {x_T, y_T, free, free, free, free}
- Initial state= {x0,y0,θ0, v_{x0},v_{y0},w0}={0,0,0,0,0}
- L = 0.2 m (distance between wheels sharing an axle)
- W = 0.1 (distance between wheel axes)
- σ^2 =(½*max (L,W) + Clearance Distance)^2/ln (2)= 6 m^2
- Vst = 3. m/s
- Integration time: 0.1s

The following obstacles where set:

x_H_Obs= [30 30 20 20 20 20 20 20 20 20 20];

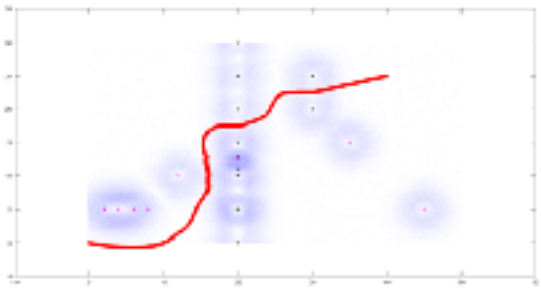
y_H_Obs= [25 20 30 25 20 15 10 5 0 11 13];

x_S_Obs= [+2 +4 +6 +8 +20 +12 +35 +20 +45];

y_S_Obs= [+5 +5 +5 +5 +5 +10 +15 +12.5 +5];

The following scaling constants were used for the guidance field: S0=H0=1.0 and T0=.01 (target field scaling constant). The target state is marked in green, the guidance field for the first step is marked in blue, the hard obstacles are marked in black, the soft obstacles are marked in magenta and the selected trajectory in red.

Figure 4: Simulation results in Matlab (sparse)



Additional runs were done for dense environments where the algorithm was also able to operate as expected. We need to stress that this is a local algorithm that was designed for sparse environments.

The algorithm is not complete, because it does not have memory or learning abilities. Therefore, it can get trapped in a loop when no solution is made available. It also does not ensure optimal paths because it operates real time and makes its decisions based solely on the best information available at the moment. This algorithm was designed to be deployed in air vessel, acting as a security feature to prevent collisions with obstacles. The choice of the ground robot for this project was done so that we could have a proof of concept without worrying with the robot's authority of control.

Figure 5: Simulation in Matlab (dense)

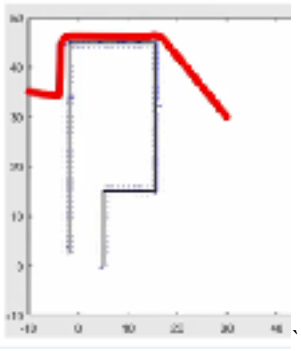


Figure 6: Simulation in Matlab: failure to return error on a path to narrow to cross

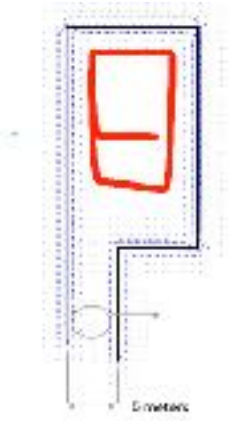


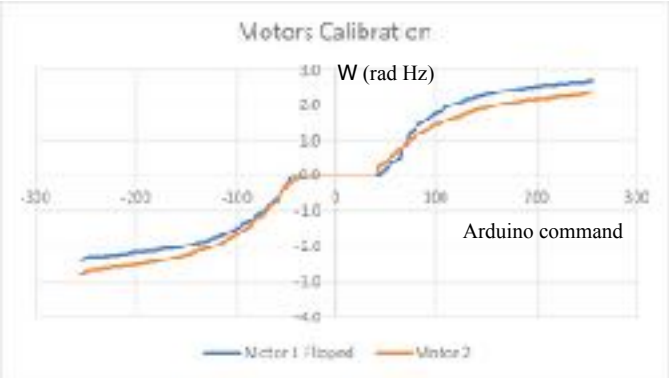
Figure 7: Simulation in Matlab: failure to find the most optimal path given that the first selected rotation (counter-clock) was not the optimal choice

I. Results obtained: Motor calibration

We have finalized the motors calibration with and without load and it showed that the not only the two motors behave differently, they have a dead energy band for low voltage inputs from the Arduino (255 corresponds to +5 V and 255 to – 5V that are then converted to +12 to – 12 V by the motor drive board).

Each of the motor brunches was fitted with a third level polynomial with chi^2 > 0.992 that will be used in converting the motors input so that they behave exactly the same way when given the same input from the algorithm.

Figure 5: Motors calibration without load



The final calibration can be found in the code uploaded together with this assignment

J. Results obtained: Algorithm deployment in real robot

After all the code was integrated in the robot, we initiated sub-system testing. The first tests where done with the IMU/ Localization boards to check that we could measure the state of the robot and got consistent measurements for the target point we had defined (x_t=2 m, y_t=2m).

IV. CHALLENGES AND FUTURE WORK

The first challenge that we faced was performing detailed trade studies across the different sub-systems in the market to ensure that we had a good performance at a fair price, in particular because we are financing the project ourselves.

The second challenge was fine tuning the obstacle avoidance algorithm (in simulation) to ignore objects when they become irrelevant while still ensuring the robot always picked a "smart" direction to circle the robot while transversing a sparse environment. This implied the need to seamlessly switch between the sparse and dense modes without generating instabilities or trapping the robot.

Finally, when we deployed all the code in the robot, we had several integration problems we had not anticipated (overflow in the Arduino, wrong heading axes built for the LIDAR data, misalignment in the initial heading of the IMU and the localization modules). This reminded us that simulation is much easier than real life!

We got a good video of the target guidance, and also, another good video for obstacle avoidance without target guidance (kind of a random walker). Both of them uploaded at: https://drive.google.com/open?id=1tYahK1XoO-8VFFP2wF5T4Tc8tDGrqXhP

Future work during the summer

For the purposes of this paper the algorithm was defined over a 2D plane, but it can be extended in future work to 3D environments. During the summer, Nihar and Joao plan to extend the algorithm to a quadcopter during their internship at Near Earth Autonomy. Instead of a target the code will accept a pilot input. The obstacles will equally come from a LIDAR scanner (this time a Velodyne VPL - 16).

This internship will leverage on the MRSD project work where a helicopter pilot aid was developed combining sound and visual warnings with an emergency break feature (that stops when obstacles show up in the quadcopter path, while in this project we are working with an algorithm that would instead circle the obstacle).

ACKNOWLEDGMENTS

We would like to thank CMU professors Matthew Travers (project advisor) and Oliver Kroemer (Robot Autonomy for which class this project was conceived) for the freedom they have allowed us in designing this project and the tips they have provided along the way.

REFERENCES

1. "Mixed Initiative Control of a Roadable Air Vehicle for Non-Pilots", Michael C. Dorneich1, Emmanuel Letsu-Dake, Sanjiv Singh, Sebastian Scherer, Lyle Chamberlain, Marcel Bergerman, Journal of Human-Robot Interaction, Vol. 4, No. 3, 2015,
2. "Development of the Side Component of the Transit Integrated Collision Warning System", Aaron Steinfeld, David Duggins, Jay Gowdy, John Kozar, Robert MacLachlan, Christoph Mertz, Arne Suppe, Charles Thorpe, Chieh-Chih Wang - 2004 IEEE Intelligent Transportation Systems Conference, Washington, D.C., USA, October 3-6, 2004
3. "Automotive HUDs: The Overlooked Safety Issues", Daniel R. Tufano, Human Factors: The Journal of the Human Factors and Ergonomics Society 1997 39: 303
4. "Motor Schema – Based Mobile Robot Navigation", Ronald C. Arkin, Proceedings of the IEEE International Conference on Robotics and Automation, April 1987
5. "AuRa: Principles and Practice in Review", Ronald C. Arkin and Tucker Balch, algorithm implemented in the winner of the 1994 AAAI Mobile Robots Competition
6. "Robot Modelling and Control", John Wiley and Sons, January 2016
7. "Dynamic Guidance of Gliders in Planetary Atmospheres", Rui Dilão and Joao Fonseca, Journal of Aerospace Engineering, April 2015
8. "Dynamic guidance of orbiter gliders: Alignment, final approach and landing", Rui Dilão and Joao Fonseca, under peer revision at CEAS Space Journal in 2018.
9. "Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework", Rached Dhaouadi and Ahmad Abu Hatab, Advances in Robotics & Automation, June 2013