Obstacle Detection and Traversal with Noisy Range Sensors

Mason U'Ren* and Jason T. Isaacs*

Abstract— **The problem of obstacle detection and avoidance for wheeled mobile rovers with noisy ultrasound range sensors is discussed. We propose a path planning algorithm, split into detection and traversal behaviors, which overcome the problems associated with noisy range sensors. Said methods first monitor and verify perceived objects then gerate appropriate control inputs. Robust detection schemes afford accurate object recognition which lends themself to smoother and quicker traversal. The proposed approach was applied to the problem of multi-agent foraging where individual rovers must avoid each other, as well as static obstacles. We provide examples of various obstacle encounters and traversal scenarios using high fidelity simulations built in the ROS and Gazebo environments. We aim to analyze the influence of monitoring and verifying sonar detections as well as evaluate new path planning techniques.**

I. Introduction

Multi-Agent Systems (MAS), an abstraction of swarm robotic systems, use a distributive model, to effect decisions and controls on agents, which tend toward a common inter-agent goal or collective swarm behavior. Applications of MAS can be seen in autonomous farming, search-and-rescue drones, and interplanetary resource foraging. Fundamental to system cohesion, agents (rovers) leverage sensorial feedback, of relative surroundings, by generating path planning manuevers, modeled by detection and traversal behaviors, where detection characterizes object recognition and traversal describes reactionary movement. Due to inheret noise, specifically within sonar range sensors, detection accuracy is limited, reducing confidence in perceived relationship between rover and object. This work seeks to mitigate this observed noise while improving upon current obstacle path planning techniques.

Path planning in the presence of obstacles for mobile robotics has a history spanning several decades. Due to space limitations, only a subset of these results are discussed in this paper. Of those surveyed, the algorithms differ in the way they avoid obstacles, either through goal orientation or edge detection.

Artificial Potential Field (APF) is a goal oriented algorithm based on the principle of a potential field in which rovers and obstacles act as positive charges, and goals act as negative charges [1]. Although a simple idea, APF has a sensitivity to local minima that usually arises from the symmetry in the environment and within concave obstacles [2]. Other drawbacks that plague APF are summarized in [3].

Proposed by Sezer et al. in [4], Follow the Gap Method (FGM) is a three stage, goal oriented algorithm, constructed by a gap array around the rover that is compared against a gap threshold, when rovers and obstacles are considered circular objects. If the calculated gap is large enough, the rover will proceed along the gaps angle bisector creating a smooth trajectory to it's target location, but it is still not immune to local minima.

Another approach, first suggested by Borenstein et al. [5], Virtual Force Field (VFF) combines the APF method with a two dimensional certainty grid defined by a cartesian histogram plane of virtually represented obstacles. This grid creates a sensitivity to certainty values in large clusters (obstacle boundaries), which function as a repulsive force. From there, APF is applied to the histogram grid, therefore the problems that hindered APF are native to VFF.

The Vector Field Histogram (VFH) method uses a two step data reduction process that models the feedback with three levels of abstraction [6]. Each level functions as a data reduction step that allows the rover to supersede blurry and inaccurate sonar feedback by expanding on the certainty values inherent to VFF. Although resistant to noisy range sensors, this method does not guarantee convergence within critical obstacles and is computationally expensive [7].

Among the earliest methods, the Bug family of algorithms are fundamental and complete algorithms since they allow the rover to reach their target point if it lies within a given space [7]. Relying on edge detection, there are three different flavors to this algorithm, Bug-1 [8], Bug-2 [8], and Dist-Bug [9], [10] which respectively improve on each other.

Bug-1 and Bug-2 algorithms are among the earliest and most elementary sensor based planners with provable guarantees [11]. Both function by creating a leaving point but differ in the calculation. Where Bug-1 is dependent on a full traversal to identify the ideal coordinates, Bug-2 compares slopes of current pose and obstacle initialization pose with the target waypoint [7]. Application of this algorithm is conducted on single-wing wholly rotating air vehicle in [12]. Although Bug-2 typically offers an improvement on Bug-1, there are special cases where the path may increase in length [13].

Dist-Bug is considered the most improved, since after detection it compares the current rover pose with the previous, checking if tangential distance to the target is increasing. Using range data, the algorithm defines a new leaving condition which allows the rover to abandon obstacle boundaries as soon as global convergence is guaranteed, based on free range to the target [10].

A common drawback to edge detection methods is their sensitivity to sensor accuracy. Ultrasonic sensors present many shortcomings with respect to poor directionality, fre-

*M. U'Ren and J.T. Isaacs are with the Department of Computer Science, California State University, Channel Islands, mason.uren600@myci.csuci.edu, jason.isaacs@csuci.edu

quent misreadings, and specular reflections [5], where any one of these errors can cause the method to determine the existence of an edge at an incorrect location. Our proposed version of the Bug algorithm overcomes this sensitivity to noisy range measurements. The Bug family is a logical avoidance choice since it allows us to isolate the algorithm's results by virtue of immunity to local minima and path to target assurance, if one exists. Any conclusions can then be readily justified on this basis. Instead of using a direct Bug algorithm, we propose a new version that takes in current rover orientation to target location, rather than distance, to decide exiting points. This should not effect results since the biggest advantage the Bug Algorithms have is their safe traversal of obstacles which remains the same. Our main contribution is a new detection algorithm that allows the rover to declare with confidence that an obstacle has been encountered without sacrificing goal convergence.

The proposed obstacle detection and avoidance approach was developed and validated on the NASA Swarmathon 2017 [14] swarm robotics platform. The Swarmathon platform was chosen due to the fact that it is publicly available, and the robotic tasks are sufficiently rich as to illustrate the benefits of this approach. The platform comes with both physical rovers and a matching simulator for virtual testing. Our algorithms are implemented in C++ using the Robot Operating System (ROS) [15] framework and the Gazebo plugin for the Graphical User Interface (GUI). The rovers are equipped with three noisy sonar range sensors, and a global positioning system receiver. The GPS receiver provides feedback for general navigation, and the sonar sensors are used to detect obstacles

In summary, the main contribution of this work is to provide obstacle detection and avoidance algorithms that allow the mobile robots to perform a variety of swarm robotic tasks in the presence of obstacles. To accomplish this task the obstacle detection algorithm must distinguish between both static obstacles and mobile obstacles.

The remainder of this paper is organized as follows. Section II provides a mathematical description of the sense and avoid with noisy range sensor problem, which is followed in Section III by an explanation of the proposed approach. Next, experimental and simulation results are examined in Section IV, followed in Section V by the final conclusions and directions for future work.

II. Problem Statement

The forthcoming collision detection and obstacle transversal algorithms are motivated by the desire to use swarms of small inexpensive rovers to explore unknown environments in search of resources. Each rover is assumed to be a small, autonomous, skid-steer, robotic vehicle driven by four independently powered wheels. Each rover is equipped with three sonar sensors, a webcam, an inertial measurement unit (IMU), wheel odometry sensors, and a global positioning system (GPS) receiver. Rotating the tires at different speeds creates skid-steer which gives the rover navigational control while maneuvering within a 2-dimensional xy-plane. Hence, if the speed of the right tires are greater than the left, the rover will steer left and vice-versa. Although the following explanation details the use of a skid-steer rover, the proposed algorithms can be easily applied to general wheeled mobile rovers equipped with noisy range sensors.

The state of a skid-steer vehicle x can be represented by the triplet $(x_d, y_d, \theta) \in SE(2)$, where $(x_d, y_d) \in \mathbb{R}^2$ describe the position of the vehicle center of gravity and $\theta \in \mathbb{S}^1$ represents the orientation. Vehicle control input $u \in \mathbb{R}^2$ is modeled as $(v, \omega) \in \mathbb{R}^2$, where v is the forward velocity of the vehicle center of gravity and ω is the rate of change in vehicle orientation. Motion of the vehicle evolves according to the following kinematic equation:

$$\dot{x} = \begin{bmatrix} \dot{x_d} \\ \dot{y_d} \\ \dot{\theta} \end{bmatrix} = f(x,\ u) = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix} \tag{1}$$

For more details on skid-steering kinematics the authors refer you to [16].

Our goal is to then find a safe trajectory from the rovers present location to a goal location, in the presence of obstacles, where a safe trajectory is achieved by an intelligent $go-to\ goal$ algorithm that navigates without error to the goal waypoint. We then assume that a $go-to\ goal$ control algorithm exists that, in the absence of obstacles, will guarantee rover traversal to the goal waypoint [16]. This controller is modeled by:

$$u = h(x_c,\ x_g) \tag{2}$$

where $x_c \in SE(2)$ describes the rover's current pose and $x_g \in SE(2)$ describes the rover's goal pose.

When in the presence of obstacles, we rely on the three low-cost, sonar sensors to provide feedback to our algorithm. Each sensor has a maximum range of R_{max} in a radial pattern of β radians as shown in Figure 1. A sensor is composed of a sending and receiving node, where the sender emits an ultrasonic ping while the receiver listens for the return of the sound wave. If the receiver hears the return of the ultrasonic wave, it implies an object is within sonar range, at which point a distance calculation is made based upon the speed of sound within the medium of travel and the time between pings. We can model this equation with:

$$D_\alpha = S(v_c,\ \triangle t,\ n) \tag{3}$$

where D_α is the rover's measured distance from the obstacle, $\alpha \in \{l,\ c,\ r\}$ differentiates between the $left,\ center$, and $right$ sonars respectively, v_c is the speed of sound in air, $\triangle t$ is the difference in time between sending and receiving pings, and n is a unknown disturbance term. If the value of D_α falls below a specified threshold distance from the rover, an obstacle detection is flagged thereby triggering an obstacle avoidance maneuver. Due to the noise term n in (3), it is possible for the the rover to chatter between the $go-to\ goal$ and $avoid\ obstacle$ controllers.

The goal then becomes to determine an $avoid\ obstacle$ control algorithm that successfully traverses obstacles, and an

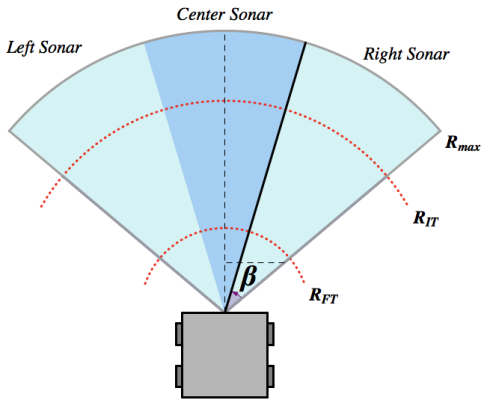

Fig. 1: *Rover decision thresholds.*

obstacle detection algorithm that allows switching between the $go-to\ goal$ and $avoid\ obstacle$ controllers without the chattering that can occur due to noisy range measurements.

III. Algorithm Implementation

Design of our avoidance algorithm can then be broken down into two classes: 1) accurately detecting obstacles, and 2) planning a safe traversal path from the object initialization location to our goal waypoint.

A. Detection

If the rover is to be driven by the $go-to\ goal$ implementation of (2) in the presence of obstacles, we must rely on the sonar sensors to accurately detect objects in the area in which it's traversing. Large errors from the sonar sensors, specifically at greater distances, can cause the rover to prematurely switch to obstacle avoidance mode, thus outlier detection must be performed on the data fed back from the sonar sensors in (3). We introduce a double threshold sonar data monitoring system (DTM) to perform the outlier detection. The idea is to have two separate distance thresholds that are chosen based on empirical measurements of the accuracy of the sonar sensors. These threshold values for R_{IT} and R_{FT} were determined from the experiments described in Subsection IV, the results of which can be found in Table I. The rover starts monitoring possible obstacles once an upper threshold, R_{IT}, is met and only then declares them obstacles once a second lower threshold, R_{FT}, is met.

Once within our initial threshold, the rover needs a way of actively monitoring the distances returned from the sonar sensors in (3). By sequentially adding new readings to respective first-in first-out (FIFO) vector arrays, denoted as V_l, V_c, and V_r, measurements can be tracked for inconsistencies, where V_{max} dictates their capacity.

Each vector is then checked against two conditions: 1) the values must be acceptable, and 2) one of the recorded distances must cross the minimum threshold, R_{FT}. Algorithm 1 describes this process where C_{val} is the current vector value, P_{val} is the previous vector value, and ρ is the combined head on collision minimum reaction distance.

Distance thresholding compares current and previous values against ρ to ultimately verify vector admissibility. Then

Algorithm 1 *Vector Iterator*: Iterates through a vector of range measurements and returns minimum value.

Input: $V \in \mathbb{R}^{V_{max}}$.
Output: S: the smallest value in the input vector or (-1) if vector contains inadmissible values.

```
if V ∈ ∅ then
    S = −1
else
    for d_i ∈ V s.t. i ∈ ℕ, 1 ≤ i ≤ V_max do
        if i = 1 then
            C_val = d_i
        else
            P_val = C_val
            C_val = d_i
            if |P_val − C_val| ≤ ρ then
                S = C_val
            else
                S = −1
                break
if S < 0 then
    Clear V
return S, V
```

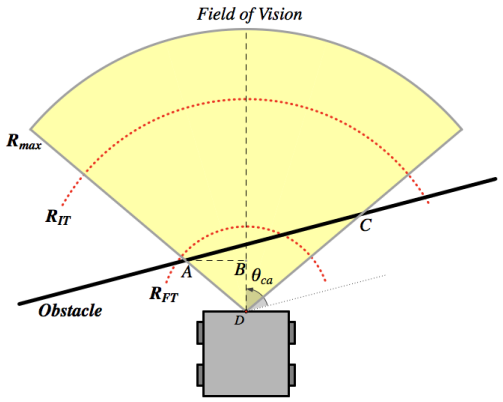

Fig. 2: *Sonar ranges pictured are limited to 2m since monitoring does not occur outside of 1.5m*

with each iteration of the main control loop Algorithm 2: $Detection\ Function$ executes to decide whether or not to take action on the latest sonar values. If no sonar measurements fall within the initial threshold R_{IT}, then no action is taken. Although once we're confident with our readings, $D_\alpha \leq R_{IT} : \alpha \in \{l, c, r\}$, we populate respective vectors with each iteration, until it reaches the specified capacity where it is either validated or dumped. The final check before obstacle declaration occurs, verifies that a vector, V_α, and it's corresponding minimum reading, S_α, are not empty and below the R_{FT} respectively, otherwise clear each vector and repopulate. At this point an obstacle has been detected and a correction angle θ_{ca} needs to be generated based on the amount of acceptable vectors.

Depending on how many vectors are being considered there are two possible θ_{ca} calculators, $wall$ and $angle\_calc$.

Algorithm 2 *Detection Function*: decides whether to take action on values from $vector\ iterator$

Input: $\mathcal{D} = \{D_l, D_c, D_r\}$ from equation 3.
Output: θ_{ca}: the rover's obstacle correction angle;
S_T: left sonar value used for traversing;

1: **if** $(D_l > R_{IT})$ & $(D_c > R_{IT})$ & $(D_r > R_{IT})$ **then**
2: No action taken.
3: **else**
4: $V_l \leftarrow D_l$
5: $V_c \leftarrow D_c$
6: $V_r \leftarrow D_r$
7: **if** $(|V_l| = V_{max})$ | $(|V_c| = V_{max})$ | $(|V_c| = V_m)$ **then**
8: $S_l = Vector\ Iterator(V_l)$
9: $S_c = Vector\ Iterator(V_c)$
10: $S_r = Vector\ Iterator(V_r)$
11: **if** $\exists \alpha \in \{l, c, r\} : \{(S_\alpha \leq R_{FT}) \& (V_\alpha \neq \emptyset)\}$ **then**
12: **if** $\{V_l, V_c,\ V_r\} \notin \emptyset$ **then**
13: **if** $S_l < S_r$ **then**
14: $\theta_{ca} = wall(S_l,\ S_c,\ S_r)$
15: **else**
16: $\theta_{ca} = wall(S_r,\ S_c,\ S_l)$
17: **if** $\theta_{ca} = \pi$ **then**
18: **if** $S_l < S_r$ **then**
19: $\theta_{ca} = angle\_calc(S_l,\ S_c)$
20: **else**
21: $\theta_{ca} = angle\_calc(S_r,\ S_c)$
22: **else**
23: **if** $\{V_l,\ V_c\} \notin \emptyset$ **then**
24: $\theta_{ca} = angle\_calc(S_l,\ S_c)$
25: **else if** $\{V_r,\ V_c\} \notin \emptyset$ **then**
26: $\theta_{ca} = angle\_calc(S_r,\ S_c)$
27: **else if** $V_l \notin \emptyset$ and $\{V_c,\ V_r\} \in \emptyset$ **then**
28: $\theta_{ca} = angle\_calc(S_l,\ -1)$
29: **else if** $V_c \notin \emptyset$ and $\{V_l,\ V_r\} \in \emptyset$ **then**
30: $\theta_{ca} = angle\_calc(S_c,\ -1)$
31: **else if** $V_r \notin \emptyset$ and $\{V_l,\ V_c\} \in \emptyset$ **then**
32: $\theta_{ca} = angle\_calc(S_r,\ -1)$
33: $S_T = S_i$
34: **else**
35: Clear V_α
36: **return** $\theta_{ca},\ S_T$

If all three vectors are considered valid, it implies that we have three acceptable minimum sonar readings ($S_l,\ S_c,\ S_r$), at which point we need to determine which exterior value is the smallest. Once we have prepped the variables, we pass them to the $wall$ function, which using the law of sines within the triangle $\triangle ADB$, calculates θ_{ca}, as shown in Figure 2. Taking the given value A and calculating B, the result is compared against the expected value C, with some degrees of freedom. It then flips the orientation of the algorithm, given B and C, and verifies that A is of expected value. If so, the original θ_{ca} is accepted and passed to traversal, but if not, θ_{ca} is flagged with π (Line 17 in Algorithm 2), and the smallest two vector values are passed to $angle\_calc$. The value of π is chosen to be the flag since the correction angle is bounded by 0 and π, not inclusive, which is further discussed in subsection III-B. $Angle\_calc$, uses similar techniques, but lacks the ability to verify results. If given two vectors, either $\{V_l,\ V_c\}$ or $\{V_r,\ V_c\}$, it calculates θ_{ca} using given value A and computed value B, but lacks the knowledge of the expected value of C. Unique to this function, when the rover is limited to one valid vector, it uses a "static avoidance" angle, determined by the rover's field of vision. If field of vision breaks down into three virtual sonar cones (Figure 1), we cannot assume each sensor views an equal piece of the rover's vision, since noise causes overlap and expantion of propogating sonar waves. But by using the approximate angle bisector of either left or right virtual sonar cone, we overcome overlap and navigate correctly to traversal. It is important to note that there is not a case, that the two vectors considered would only be $\{V_c,\ V_r\}$, due to the amount of overlap between individual sonar cones. Once a vector has been accepted and flagged for examination, it is compared with the remaining established vectors, dictating how our correction angle is calculated. The correction angle then intends to orient the rover into a clear traversal path parallel to the obstacle.

B. Distinction & Traversal

Building on the safe traversal guaranteed by Bug Algorithms, we propose a hybrid bug algorithm (HBA), that intelligently distinguishes and traverses objects. But before traversal can begin, the rover needs to verify that the object is truly an obstacle and not a resource. Ultimately, $obstacle\ checking$ is done the same way regardless of vector admissibility, but with a slight difference between how the checking angle is calculated, where the checking angle is the angle of rotation needed to face the object detected. When more than one vector is accepted, the algorithm uses the complement of θ_{ca} to rotate in place toward the object for checking. When only working with one vector, the rover is rotated to face to the object. Depending on the state preceding the Obstacle Avoidance State, if resources are found, the rover transitions out of the current state and proceeds to pick-up (search state). Unless the rover is already carrying a resource (delivery state), in which case it would treat both, objects and resources, as obstacles. In either case, once an object is declared an obstacle the rover corrects back to a parallel traversal path. It is important to note that regardless of which side the rover detects obstacles, the correction angle always preps the rover for counter-clockwise rotation. Since the rover only traverses in this manner, we are able to make the above assumption about the correction angle ($0 < \theta_{ca} < \pi$). If the rover's orientation lives on or outside of these bounds, it is driving parallel to the obstacle or away from the obstacle. If the rover is driving away, the obstacle has been passed, otherwise the only way the rover arrives parallel to an object is when it has previously been detected and traversal has begun.

Traversal is then built upon three crucial concepts, $Initialization$, $Traversing$, and $Checking$. During traver-

sal, the rover uses a technique called $wall\ following$, which utilizes the left sonar vector's minimum value returned from Algorithm 2, S_T, to gauge relative current distance from the obstacle it's traversing. This is achieved by, using the dynamic drive system mentioned in (1), while bounding S_T, thus creating a virtual road that allows for smooth traversal. This can be seen in Algorithm 3, $Traversal\ Setup$, where $R_{(x,y,\theta)}$ refers to the rover's current pose, T_i corresponds to the rover's initialization, $T_\perp$ is the ideal perpendicular distance from rover and edge, and x_g is the goal waypoint in an xy-plane.

Algorithm 3 *Traversal Setup*: adds new waypoints based on rover's current pose

Input: S_T: current tangential distance to obstacle
Output: x_g: set new effective waypoint specific to avoidance

1: **if** $S_T < T_{min}$ and $T_i = False$ **then**
2: $T_{(x,y)} = \frac{1}{30}\sum_{n=1}^{30} R_{(x,y)}$
3: Set $T_i = True$
4: **else if** $S_T < 0.2m$ **then**
5: $x_g = rotate\_right(0.35\ rads)$
6: **else**
7: **if** $S_T < T_{min}$ **then**
8: $\theta_{ca} = \frac{-0.08}{S_T}$
9: **else if** $S_T > T_{max}$ **then**
10: $\theta_{ca} = \frac{0.08}{S_T}$
11: $x = T_\perp \times \cos(R_\theta + \theta_{ca}) + R_x$
12: $y = T_\perp \times \sin(R_\theta + \theta ca) + R_y$
13: $x_g = (x, y)$
14: **return** x_g

Algorithm 4 *Traversing*: monitors the rovers current waypoint and resets x_g only if resource is found, full traversal has been made, x_g is visible on the right side of the rover, or rover is driving outside of boundaries

Input: x_g: a goal waypoint to drive to
Output: $x_g \in \emptyset$

1: **if** $W_\theta \geq 0$ **then**
2: $\delta = 2\pi - W_\theta$
3: **else**
4: $\delta = |W_\theta|$
5: **if** $cube\ detected$ **then**
6: Set $x_g = \emptyset$
7: **else if** $R_{(x,y} = W_{(x,y)}$ **then**
8: Goal waypoint has been reached while traversing.
9: Set $x_g = \emptyset$
10: **else if** $\frac{\pi}{4} \leq W_\theta \leq \frac{\pi}{2}$ **then**
11: Goal waypoint is to the right of rover's orientation.
12: Set $x_g = \emptyset$
13: **else if** $T_i = True$ **then**
14: **if** $R_{(x,y,\theta)} \approx T_{(x,y)}$ **then**
15: Rover has done a full traversal.
16: Set $x_g = \emptyset$
17: **if** $S_T = 0$ **then**
18: Rover is at an exterior corner.
19: Set $x_g = \emptyset$
20: **else if** $S_T < 0.2$ or $\{S_c,\ S_r\} \notin \emptyset$ **then**
21: Set $x_g = \emptyset$
22: **else**
23: **if** $S_T < T_{min}$ or $S_T > T_{max}$ **then**
24: Set $x_g = \emptyset$
25: **return** x_g

Once first encountered, the rover records its initial point of contact, $T_{(x,y)}$, on the obstacle, which is a necessary step when determining if a given Search waypoint is unreachable. If while traversing, such a waypoint is marked "unreachable", the rover will remove the waypoint, exit Obstacle Avoidance State, and proceed to the next waypoint. Otherwise, once $Initialization$ is finished ($T_i = True$), the rover will begin traversing the obstacle in a counter-clockwise manner. Both drive systems use dynamic corrections, except traversal is dependent on how far the rover exceeds lower and upper bounds, triggering a new x_g passed to Algorithm 4 for monitoring. The $Traversal\ Setup$ and $Traversing$ algorithms work hand-in-hand within the state machine. $Traversal\ Setup$ is the function call that creates x_g based on the various checks made, while $Traversing$ serves as a monitor of the waypoints. If the x_g passes all the checks in place then no action is taken and the state machine will continue to loop until x_g is reached. But if action is needed the algorithm will clear x_g and pass the decision back to $Traversal\ Setup$ in order to create a new waypoint based on sensorial criteria.

This method of $Traversing$ is ideal for generally smooth obstacles, but becomes more complicated with sharper, more realistic objects. When dealing with sharper obstacles, the two challenges that present themselves are how to get around local minima with sharp interior angles and navigating around exterior angles with little to no sensor input. We can manage navigation of interior angles by monitoring our remaining two sonar vectors. Corrections are being made to the rover's path periodically based on the S_T, in an attempt to keep it relatively close to the obstacle, but if any detections are made by the other vectors, (S_c, S_r), a static correction angle of $0.35\ rads$ (Line 5, Algorithm 3), in the clockwise direction, is used to navigate the interior corner. During rotation, traversal checks are then continually made on the left vector, ensuring that once this has exceeded the traversal's lower bound, normal traversal will resume. While interior angles are traversed by monitoring all sensor data, the rover loses all sensorial information when encountering exterior angles. This means that the rover will effectively "fly blind" for a specified period of time or declare the obstacle passed. Further complicating the issue, the rover will lose it's left vector data approximately $0.38m$ away from the apex of the corner. Thus, the rover needs to drive forward a static distance before it begins to make the turn, otherwise it will enter an endless loop of re-recognizing the obstacle, then losing it as it turns away in an attempt to orient itself

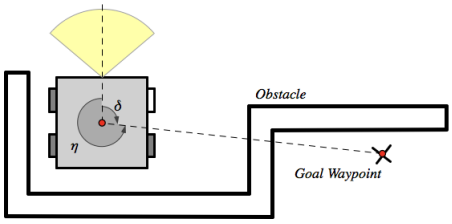

Fig. 3: *Calculated correction angle η while traversing obstacle.*

down the traversal path. At this point, Algorithm 4 would not transition back to Algorithm 3 but would instead look to Algorithm 5 to create a series of waypoints to properly navigate the corner. $Check$ wraps up this notion of dynamic

Algorithm 5 *Check*: rotates the rover around exterior corners while monitoring orientation to x_g

Input: η: the goal angle calculated in a counter clockwise manner.

Output: x_g: a new traversal waypoint or leaving waypoint

$iter = \theta_{ca} \times 3$
$x_g = (x, y)$ s.t. $\theta_{ca} = R_\theta$ and $\sqrt{x^2 + y^2} = 0.3m$
for $i < iter$ s.t $\{i \geq 0\} \in \mathbb{N}$ **do**
    $x_g = rotate\_left(\frac{\beta}{iter})$
    $x_g = (x, y)$ s.t. $\theta_{ca} = R_\theta$ and $\sqrt{x^2 + y^2} = 0.025m$
$x_g = rotate\_left(0.3\ rads)$
return x_g

driving by first statically driving forward $0.3\ m$, placing the rover roughly at the corner's apex, then dynamically creating several incremental waypoints to form an arc around the corner. To achieve proper cornering, the correction angle, η in Figure 3, is re-calculated through the counter clockwise difference between current rover orientation and tangential distance between rover and goal waypoint. From the illustration it can be deduced that if the correction angle were calculated in a clockwise manner, the rover would rotate right, exit Obstacle Avoidance State, then re-enter within the local minima of the obstacle, at which point the rover would be in an endless loop. During dynamic cornering, Algorithm 4 monitors the left vector again to verify when the lower bound has again been crossed, at which point normal traversal resumes.

Although traversal and initialization guide the rover around the obstacle, the key to obstacle avoidance is knowing when to properly exit. This idea is a subset of traversal and is achieved through checking a combination of four conditions: 1) has the rover traversed the entire obstacle, 2) has the rover driven over the goal waypoint during traversal, 3) can the rover see a clear path to the waypoint, and 4) has the rover found a resource during traversal, which due to space limitations, the later will not be discussed.

The first condition was briefly discussed above so we'll skip to the second condition. When in Search State the rover is driving to a set of fixed waypoints that are laid out by the search algorithm, but once Obstacle Avoidance State has been declared, a new set of waypoints are generated specific to avoidance. Thus the rover needs to remember the goal waypoint while in avoidance, but since the GPS on the rover is not reliable within $1m$, we treat the goal waypoint as a boundary rather than a singleton. Then while traversing the obstacle, if the GPS marks the rover within the boundary, the waypoint is marked found and avoidance is exited.

Finally, while traversing a smooth edge or exterior corner, the rover should always be judging itself in relation to the goal waypoint, deciding whether it's acceptable to leave traversal. For exterior angles, this occurs when the correction angle is less than the angle needed to re-register the obstacle with the left sonar vector, at which point the avoidance would exit. But, there are a finite number of exterior corners on any given obstacle, thus while traversing smooth edges, the rover constantly calculates the inverse of η, angle δ shown in Figure 3. If δ ever falls between lower and upper exiting bounds, $\frac{\pi}{4}$ and $\frac{\pi}{2}$ the goal waypoint exists to the right of the rover, and avoidance would be exited. This is allowable since even if the rover is within a local minima, the waypoint is such that it would cause the rover to cut the minima shallow, pick up the obstacle on the other side, then transition back out of avoidance when appropriate. Thus the rover can essentially multitask searching for waypoints, traversing obstacles, and looking for resource (although not discussed).

IV. Results

A. Experimental Results

A common problem when working with ultrasonic range sensors is sensitivity to noise, whether from frequency misreadings or specular reflections. Isolating the direct cause proved difficult, but experimental results in Table I show that any reading beyond $1.5m$ is uncertain. At $1.5m$ standard deviation drops to $\approx 6cm$ and improves as relative distance decreases. It is important to note that although the measured mean values are not equivalent to actual distances, their variance remains minimal, hence although not accurate, they are precise and admissible.

Sonar Accuracy					
Actual Distance	Mean	Max	Min	STD	VAR
0.5	0.33828	0.36	0.319	0.00804	6.469×10^{-5}
1.0	0.8488	0.99	0.79	0.06686	0.00447
1.5	1.2974	1.7	1.11	0.06644	0.00441
2.0	1.3018	2.16	0.97	0.15105	0.02281

TABLE I: *sonar distance measured in meters*

Thus we empirically chose the values for R_{IT} and R_{FT}, set at $1.5m$ and $0.6m$ respectively, where R_{IT} is based on reliable feedback and R_{FT} is set at the webcam's maximum resource identification distance.

Performance of DTM was then tested against a single threshold baseline in 25 individual trials. For each trial, the rover was placed at a starting distance of $2m$ away from an obstacle, then driven forward till each algorithm flagged a seen obstacle at a percieved distance of $0.6m$. As evidenced by Table II, our double threshold implementation was

Sonar Detections Goal Flagged Distance (0.6m)					
Algorithm	Mean	Max	Min	STD	VAR
Single Theshold	1.48767	1.8415	0.40005	0.33755	0.11394
DTM	0.398399	0.4699	0.2286	0.04719	0.00222

TABLE II: *Detection distance in meters*

roughly a 73% improvement on its single threshold counter part with a mean of $0.399399m$ compared to $1.48767m$ respectively. Supported by a standard deviation of $0.04719m$ and a variance of $0.00222m$, we can see that 95% of detections occur within a $0.18876m$ of eachother. Although our mean value is below our target detection distance, these errors can be accounted for as previous discussed with Table I. Thus we can conclude that DTM is effective at minimizing detection error and consistant when detecting obstacles.

B. Simulation Results

Simulations of our bug algorithm were run in the Gazebo Simulation environment with favorable results as seen in figs. 4 to 6, with comparisons between implementations being made in Figure 7 and Table III. Within each simulation, a black triangle denotes the $start\ location$ (x_c) of the rover where the $go-to\ goal$ endpoint is marked by a black X (x_g). Obstacle interactions are characterized by a few different symbols ranging from a red or $black\ circle$ marking either the $obstacle\ seen$ decision boundary or the $obstacle\ leaving$, $obstacle\ unreachable$ judgement point respectively. The $go-to\ goal$ path is broken up into the current rover states, where $search\ state$ is displayed by a solid gray line, $obstacle\ state$ either by a an orange dashed line or cyan dotted line, and $find\ home\ state$ as a dotted black line. Subtle variations in traversal and decision points will be discussed in respective figures.

Figure 4, models the safe traversal of a critical shape (U & H), where the rover approaches said obstacle, switches states correctly, then creates avoidance waypoints that direct it in the appropriate traversal till a clear path is detected to the goal waypoint. We can see that our implementation of the $Bug\ Family$ still holds the necessary characteristics which allow it to be immune to local minima and achieve an unencumbered path to target.

Similar results are seen in Figure 5 when some $go{-}to\ goal$ waypoints reside within obstacles and are thus unreachable. On approach, the rover is attempting to reach x_{g_1} located within $obstacle_1$. After the first complete traversal, marked by the cyan dotted line, x_{g_1} is cleared from the $search\ state$ queue and the goal waypoint is update to x_{g_2}, located within $obstacle_2$. The orange dashed line now marks the second complete traversal of the object, at which point x_{g_2} is ruled as unreachable and the rover transitions to locating x_{g_3}. A brief traversal then begins marked by the solid green line until x_{g_3} is seen in the direction of a clear path, thus $obstacle\ state$ transitions back out to $search\ state$. This figure demonstrates a key characteristic where each traversal is near identical to the previous, which shows that the rover is

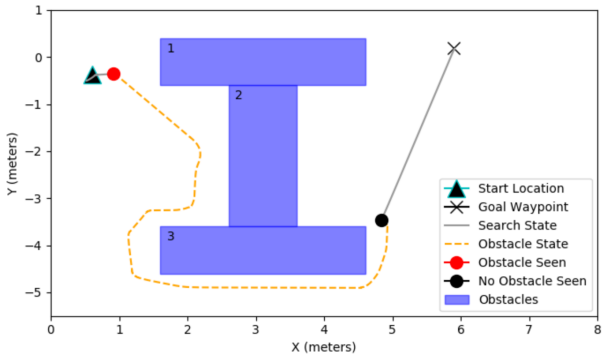

Fig. 4: *Rover odometry path when traversing through local minima*

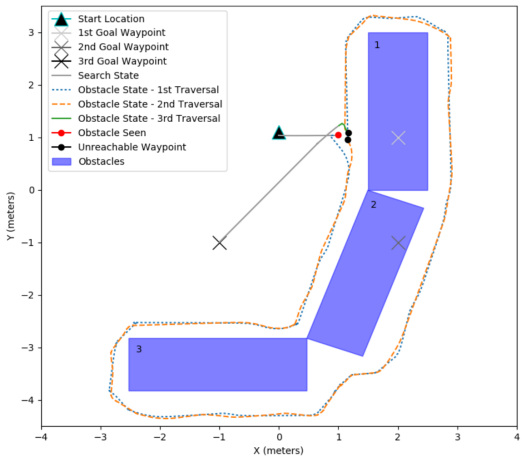

Fig. 5: *Rover odometry path when x_g is within obstacles*

making consistent decisions regardless of how it encounters the obstacle. This helps validate our algorithm by showing consistency throughout.

Figure 6 then plots the adjusted rover path when navigating between separate obstacles. While attempting to reach its goal waypoint the rover encounters $obstacle_1$ and begins traversal. While traversing the rover is constantly looking for a clear path to its goal which is seen on the lower side of the obstacle, where it transitions back into $search\ state$. Search then handles trajectory until $obstacle_2$ is seen, at which point $obstacle\ state$ again creates avoidance waypoints around the detected object. The rover exits along $obstacle_2's$ exterior corner where the goal waypoint is then achieved and the rover proceeds to the new waypoint denoted by the gray

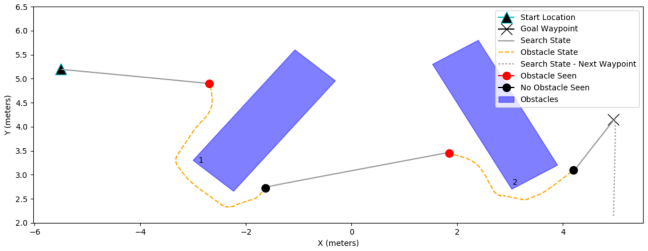

Fig. 6: *Rover odometry path with separate obstacles*

Obstacle Traversal Times					
Algorithm	Mean	Max	Min	STD	VAR
Dist-Bug	90.6838	101.4	84.6	3.99496	15.95972
HBA	72.39992	75.3	69.699	1.20592	1.45425

TABLE III: *Obstacle traversal times in simulations seconds*

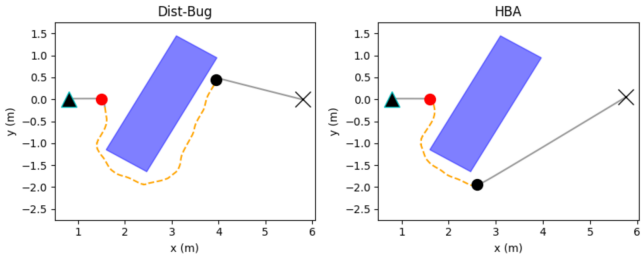

Fig. 7: *Traversal at mean recorded simulation times.*

dotted line. This figure demonstrates a key difference between our algorithm and previous $Bug\ Algorithms$, which is seen in the exit points. We can see that the exit point from $obstacle_1$ is chosen based on sight, not distances, slope, or calculated ideal leaving points. If the later were true, the exit point from $obstacle_1$ would exist further along in traversal.

Finally, assuming accurate obstacle detection, Table III compares the performance of HBA against the Dist-Bug algorithm, over 25 trials each. Using static starting/goal positions, we can see our implementation exerts $\approx 20\%$ decrease in traversal time with an average of 72.39992 simulation seconds (sim-sec), as compared to 90.6838 sim-sec. This outcome is further supported by the difference betweeen Dist-Bug$_{min} = 84.6$ sim-sec and HBA$_{max} = 75.3$ sim-sec, which suggests that the best Dist-Bug traversal was still slower than the worst HBA traversal. Illustrated by Figure 7, traversal times decrease when exit points are based on orientation to x_g rather than differences in $go-to\ goal$ distances, which tracks the rover down a wasteful path down the obstacles edge. Although, the obstacle's orientation was chosen to exagerate the benefits of our algorithm, it is important to note that reorrientation, to an angle that placed the exit point of both algorithms on the same exterior corner, would at best allow the Dist-Bug to have an equivalent traversal time. We can reach this conclusion since the Dist-Bug algorithm can not truely leave while navigating an exterior corner since it is strictly monitoring tangential distance rather than orientation to x_g.

V. Conclusion

From our results, we can conclude that after using a hybrid version of the Bug Algorithm, a known successful obstacle traversal technique, our double threshold sonar data monitoring system worked effectively. By working with a known algorithm, we can isolate and validate our results. Objects may have been registered at a distance greater than $1.5m$, but no action was taken against them. Once the relative distance between rover and object fell below R_{IT}, the values were accurately captured within our vector arrays, V_α, and then successfully analyzed for consistency since the rover did not encounter any false detections. Then if a value, S_α dropped below the R_{FT}, it was correctly checked then marked either as an obstacle or resource at which point the appropriate action was taken.

Building on this work, we are currently applying these algorithms to multi-agent systems, where multiple rovers communicate current location in combination with discovered obstacles and resources to better navigate foreign terrain. This will allow us to improve the $Bug\ Algorithm$ by training the rover to remember encountered obstacles and resources alike, thus building a virtual map of the surrounding environment. Doing this will allow the rover to calculate and plot uninhibited paths, to and from known patches of resources, while superseding the traversal of an obstacle.

References

[1] M. Zohaib, M. Pasha, R. Riaz, N. Javaid, M. Ilahi, and R. Khan, "Control strategies for mobile robot with obstacle avoidance," *arXiv preprint arXiv:1306.1144*, 2013.

[2] J. Oroko and B. Ikua, "Obstacle avoidance and path planning schemes for autonomous navigation of a mobile robot: a review," *Sustainable Research and Innovation Proceedings*, vol. 4, 2012.

[3] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proc. of the IEEE International Conference on Robotics and Automation*, 1991, pp. 1398–1404.

[4] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm:follow the gap method," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012.

[5] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[6] ——, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[7] M. Zohaib, S. M. Pasha, N. Javaid, A. Salaam, and J. Iqbal, "An improved algorithm for collision avoidance in environments having u and h shaped obstacles," *Studies in Informatics and Control*, vol. 23, no. 1, pp. 97–106, 2014.

[8] V. Lumelsky and A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," *IEEE Trans. on Automatic Control*, vol. 31, no. 11, pp. 1058–1063, 1986.

[9] E. Magid and E. Rivlin, "CAUTIOUSBUG: A competitive algorithm for sensory-based robot navigation," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2757–2762.

[10] I. Kamon and E. Rivlin, "Sensory-based motion planning with global proofs," *IEEE Trans. on Robotics and Automation*, vol. 13, no. 6, pp. 814–822, 1997.

[11] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, "Principles of robot motion: Theory, algorithms, and implementation," *The MIT Press*, vol. 1st Edition, pp. pp. 17 – 38, 2007.

[12] J. T. Isaacs, C. Magee, A. Subbaraman, F. Quitin, K. Fregene, A. R. Teel, U. Madhow, and J. P. Hespanha, "Gps-optimal micro air vehicle navigation in degraded environments," in *American Control Conference (ACC), 2014*. IEEE, 2014, pp. 1864–1871.

[13] Y. Alpaslan and P. Osman, "Performance comparison of bug algorithms for mobile robots," in *5th International Advanced Technologies Symposium (IATS09)*, vol. 13, 2009.

[14] NASA Swarmathon, "Home - NASA Swarmathon," 2017, Retrieved on 09/29/2017 from http://nasaswarmathon.com. [Online]. Available: http://nasaswarmathon.com

[15] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[16] E. Maalouf, M. Saad, and H. Saliah, "A higher level path tracking controller for a four-wheel differentially steered mobile robot," *Robotics and Autonomous Systems*, vol. 54, no. 1, pp. 23 – 33, 2006.