Obstacle Avoidance (Local Path Planning)

- The goal of the obstacle avoidance algorithms is to avoid collisions with obstacles.
- It is usually based on local map.
- Often implemented as a more or less independent task.
- However, efficient obstacle avoidance should be optimal with respect to:
  - the overall goal
  - the actual speed and kinematics of the robot
  - the on board sensors
  - the actual and future risk of collision
Obstacle Avoidance: **Bug1**

- Follow along the obstacle to avoid it
- Each encountered obstacle is one fully circled before it is left at the point closest to the goal
- Move toward the goal and repeat for any future encountered obstacle
- Advantages:
  - **No global map required**
  - **Completeness guaranteed**
- Disadvantages:
  - **Solutions are often highly suboptimal**
Obstacle Avoidance: **Bug2**

- Follow the obstacle always on the left or right side
- Leave the obstacle if the direct connection between start and goal is crossed
Practical Implementation of Bug2

- Two states of robot motion:
  1. moving toward goal \( (GOALSEEK) \)
  2. moving around contour of obstacle \( (WALLFOLLOW) \)

- Describe robot motion as function of sensor values and relative direction to goal
- Decide how to switch between these two states

```c
while (!atGoal)
{
    if (goalDist < goalThreshold)
    {
        We're at the goal! Halt.
    }
    else
    {
        forwardVel = ComputeTranslation(&sonars)
        if (robotState == GOALSEEK)
        {
            rotationVel = ComputeGoalSeekRot(goalAngle)
            if (ObstaclesInWay())
                robotState <- WALLFOLLOW
        }
        if (robotState == WALLFOLLOW)
        {
            rotationVel = ComputeRightWallFollowRot(&sonars)
            if (!ObstaclesInWay())
                robotState <- GOALSEEK
        }
    }
    robotSetVelocity(forwardVel, rotationVel)
}
```
Practical Implementation of Bug2 (con’t.)

- **ObstaclesInWay()**: is true whenever any sonar range reading in the direction of the goal (i.e., within 45° of the goal) is too short

- **ComputeTranslation()**: proportional to largest range reading in robot’s approximate forward direction
  - // Note similarity to potential field approach!
  - If minSonarFront (i.e., within 45° of the goal) < min_dist
    - o return 0
  - Else return \( \min (\max _{velocity}, \min SonarFront - \min _{dist}) \)
Practical Implementation of Bug2 (con’t.)

• For computing rotation direction and speed, popular method is:
  - Subtract left and right range readings
  - The larger the difference, the faster the robot will turn in the direction of the longer range readings

• ComputeGoalSeekRot(): // returns rotational velocity
  - if (abs(angle_to_goal)) < PI/10
    - return 0
  - else return (angle_to_goal * k) // k is a gain

• ComputeRightWallFollowRot(): // returns rotational velocity
  - if max(minRightSonar, minLeftSonar) < min_dist
    - return hard_left_turn_value  // this is for a right wall follower
  - else
    - desiredTurn = (hard_left_turn_value - minRightSonar) * 2
    - translate desiredTurn into proper range
    - return desiredTurn
Pros/Cons of Bug2

- **Pros:**
  - *Simple*
  - *Easy to understand*
  - *Popularly used*

- **Cons:**
  - *Does not take into account robot kinematics*
  - *Since it only uses most recent sensor values, it can be negatively impacted by noise*

- More complex algorithms (in the following) attempt to overcome these shortcomings
Obstacle Avoidance: Vector Field Histogram (VFH)

- Overcomes Bug2’s limitation of only using most recent sensor data by creating local map of the environment around the robot
- Local map is a small occupancy grid
  - Grid cell values equal to the probability that there is an obstacle in that cell
- This grid is populated only by relatively recent sensor data

Koren & Borenstein, ICRA 1990

Adapted from © R. Siegwart, I. Nourbakhsh
How to calculate probability that cell is occupied?

• Need sensor model to deal with uncertainty

• Let’s look at the approach for a sonar sensor …
Modeling Common Sonar Sensor (from Murphy)

Region I: Probably occupied
Region II: Probably empty
Region III: Unknown

$R = \text{maximum range}$
$\beta = \text{field of view}$
How to Convert to Numerical Values?

• Need to translate model (previous slide) to specific numerical values for each occupancy grid cell
  ➢ These values represent the probability that a cell is occupied (or empty), given a sensor scan (i.e., \( P(\text{occupied} \mid \text{sensing}) \))

• Three methods:
  ➢ Bayesian
  ➢ Dempster-Shafer Theory
  ➢ HIMM (Histogrammic in Motion Mapping)

• We’ll cover:
  ➢ Bayesian

• We won’t cover:
  ➢ Dempster-Shafer
  ➢ HIMM
Bayesian: Most popular evidential method

- Approach:
  - Convert sensor readings into probabilities
  - Combine probabilities using Bayes’ rule:
    \[
    P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}
    \]
  - That is,
    \[
    \text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Normalizing constant}}
    \]

- Pioneers of approach:
  - Elfes and Moravec at CMU in 1980s
Review: Basic Probability Theory

- Probability function:
  - Gives values from 0 to 1 indicating whether a particular event, $H$ (Hypothesis), has occurred

- For sonar sensing:
  - Experiment: Sending out acoustic wave and measuring time of flight
  - Outcome: Range reading reporting whether the region being sensed is Occupied or Empty

- Hypotheses (H) = \{Occupied, Empty\}

- Probability that H has really occurred:
  \[0 < P(H) < 1\]

- Probability that H has not occurred:
  \[1 - P(H)\]
Unconditional and Conditional Probabilities

• Unconditional probability: $P(H)$
  - “Probability of $H$”
  - Only provides a priori information
  - For example, could give the known distribution of rocks in the environment, e.g., “x% of environment is covered by rocks”
  - For robotics, unconditional probabilities are not based on sensor readings

• For robotics, we want: Conditional probability: $P(H \mid s)$
  - “Probability of $H$, given $s$” (e.g., $P(\text{Occupied} \mid s)$, or $P(\text{Empty} \mid s)$)
  - These are based on sensor readings, $s$

• Note: $P(H \mid s) + P(\text{not } H \mid s) = 1.0$
Probabilities for Occupancy Grids

• For each grid[i][j] covered by sensor scan:

  ➢ Compute $P(\text{Occupied} \mid s)$ and $P(\text{Empty} \mid s)$

• For each grid element, grid[i][j], store tuple of the two probabilities:

```c
typedef struct {
    double occupied;  // i.e., $P(\text{occupied} \mid s)$
    double empty;     // i.e., $P(\text{empty} \mid s)$
} P;

P occupancy_grid[ROWS][COLUMNS];
```
Recall: Modeling Common Sonar Sensor to get $P(s \mid H)$

Region I: Probably occupied
Region II: Probably empty
Region III: Unknown

$R = \text{maximum range}$
$\beta = \text{field of view}$
Converting Sonar Reading to Probability: Region I

- Region I:

\[
P(\text{Occupied}) = \frac{R - r}{R} + \frac{\beta - \alpha}{\beta} \times \text{Max}_{\text{occupied}}
\]

where \( r \) is distance to grid element that is being updated

\( \alpha \) is angle to grid element that is being updated

\( \text{Max}_{\text{occupied}} = \text{highest probability possible} \) (e.g., 0.98)

P(Empty) = 1.0 – P(Occupied)

The nearer the grid element to the origin of the sonar beam, the higher the belief

The closer to the acoustic axis, the higher the belief

We never know with certainty
Converting Sonar Reading to Probability: Region II

• Region II:

\[ P(\text{Empty}) = \frac{R-r}{R} + \frac{\beta-\alpha}{\beta} \]

The closer to the acoustic axis, the higher the belief

The nearer the grid element to the origin of the sonar beam, the higher the belief

\[ P(\text{Occupied}) = 1.0 - P(\text{Empty}) \]

\textit{where} r \textit{is distance to grid element being updated,} \\
\textit{\( \alpha \) is angle to grid element being updated} \\

\textit{Note that here, we allow probability of being empty to equal 1.0}
Sonar Tolerance

• Sonar range readings have resolution error

• Thus, specific reading might actually indicate range of possible values

• E.g., reading of 0.87 meters actually means within (0.82, 0.92) meters
  ➢ Therefore, tolerance in this case is 0.05 meters.

• Tolerance gives width of Region I
Tolerance in Sonar Model

Region I: Probably occupied
Region II: Probably empty
Region III: Unknown
Example: What is value of grid cell $oxed{}$? (assume tolerance = 0.5)

Which region?

$3.5 < (6.0 - 0.5) \Rightarrow \text{Region II}$

$P(\text{Empty}) = \frac{10 - 3.5}{10} + \frac{15 - 0}{15} = 0.83$

$P(\text{Occupied}) = (1 - 0.83) = 0.17$
But, not yet there – need $P(H|s)$, not $P(s|H)$

- Note that previous calculations gave: $P(s | H)$, not $P(H | s)$
- Thus, use Bayes Rule:

$$P(H | s) = \frac{P(s | H) P(H)}{P(s | H) P(H) + P(s | not H) P(not H)}$$

$$P(H | s) = \frac{P(s | Empty) P(Empty)}{P(s | Empty) P(Empty) + P(s | Occupied) P(Occupied)}$$

- $P(s | Occupied)$ and $P(s | Empty)$ are known from sensor model
- $P(Occupied)$ and $P(Empty)$ are unconditional, prior probabilities (which may or may not be known)
  
  If not known, okay to assume $P(Occupied) = P(Empty) = 0.5$
Returning to Example

• Let’s assume we’re on Mars, and we know that \( P(Occupied) = 0.75 \)
• Continuing same example for cell...

\[
P(Empty \mid s=6) = \frac{P(s \mid Empty) \cdot P(Empty)}{P(S \mid Empty) \cdot P(Empty) + P(s \mid Occupied) \cdot P(Occupied)}
\]

\[
= \frac{0.83 \times 0.25}{0.83 \times 0.25 + 0.17 \times 0.75}
\]

\[
= 0.62
\]

• \( P(Occupied \mid s=6) = 1 - P(Empty \mid s=6) = 0.38 \)

• These are the values we store in our grid cell representation
Updating with Bayes Rule

• How to fuse multiple readings obtained over time?
• First time:
  ➢ Each element of grid initialized with a priori probability of being occupied or empty
• Subsequently:
  ➢ Use Bayes’ rule iteratively
  ➢ Probability at time $t_{n-1}$ becomes prior and is combined with current observation at $t_n$ using recursive version of Bayes rule:

$$P(H | s_n) = \frac{P(s_n | H) P(H | s_{n-1})}{P(s_n | H) P(H | s_{n-1}) + P(s_n | not H) P(not H | s_{n-1})}$$
Now, back to: Vector Field Histogram (VFH)

- Environment represented in a grid (2 DOF)
  - cell values are equivalent to the probability that there is an obstacle

- Generate polar histogram:
  - angle at which obstacle is found (relative to robot’s current position)
  - probability of occupied

Adapted from © R. Siegwart, I. Nourbakhsh
Now, back to: Vector Field Histogram (VFH)

- From histogram, calculate steering direction:
  
  ➢ *Find all openings large enough for the robot to pass through*
  
  ➢ *Apply cost function $G$ to each opening*

  $$ G = a \cdot \text{target\_direction} + b \cdot \text{wheel\_orientation} + c \cdot \text{previous\_direction} $$

  where:
  
  - $\text{target\_direction} = \text{alignment of robot path with goal}$
  - $\text{wheel\_orientation} = \text{difference between new direction and current wheel orientation}$
  - $\text{previous\_direction} = \text{difference between previously selected direction and new direction}$

  ➢ *Choose the opening with lowest cost function value*
VFH Limitations

- Limitation if narrow areas (e.g. doors) have to be passed
- Local minimum might not be avoided
- Reaching of the goal cannot be guaranteed
- Dynamics of the robot not really considered

Borenstein et al.
Movie

- VFH in action on Pioneer robot (Tracking_VFH_2x)
Obstacle Avoidance: Vector Field Histogram + (VFH+)

- Accounts also in a very simplified way for the moving trajectories
  - robot can move on arcs
  - arcs take into account kinematics
  - obstacles blocking a given direction also block all the trajectories (arcs) going through this direction

Caprari et al. 2002
Obstacle Avoidance: **Basic Curvature Velocity Methods** (CVM)  

Simmons et al.

- Adding *physical constraints* from the robot and the environment on the *velocity space* \((v, \omega)\) of the robot
  - Assumption that robot is traveling on arcs \((c = \omega / v)\)
  - Constraints: 
    \[-v_{\text{max}} < v < v_{\text{max}}\]
    \[-\omega_{\text{max}} < \omega < \omega_{\text{max}}\]
  - Obstacle constraints: Obstacles are transformed in velocity space
  - Objective function used to select the optimal speed
Obstacle Avoidance: **Dynamic Window Approach**

Fox and Burgard, Brock and Khatib

- The kinematics of the robot is considered by searching a well chosen velocity space
  - velocity space -> some sort of configuration space
  - robot is assumed to move on arcs
  - ensures that the robot comes to stop before hitting an obstacle
  - objective function is chosen to select the optimal velocity

\[
O = a \cdot \text{heading}(v, \omega) + b \cdot \text{velocity}(v, \omega) + c \cdot \text{dist}(v, \omega)
\]

- **heading** = progress toward goal
- **velocity** = forward velocity of robot (encourages fast movements)
- **dist** = distance to closest obstacle in trajectory

Adapted from © R. Siegwart, I. Nourbakhsh
Obstacle Avoidance: **Global Dynamic Window Approach**

- **Global approach:**
  - *This is done by adding a minima-free function named NF1 (wave-propagation) to the objective function $O$ presented above.*
  - *Occupancy grid is updated from range measurements*