Trabalho 2 - Abordagem para Desvio de Obstáculos utilizando Robô Pioneer 3AT

Andressa Sousa da Silveira - 10/0053971
Lucília xxxxxxx xxxxxxxxx - XX/XXXXXXX
Matheus Medeiros Sarmento - 12/0053161
Rondinele Barbosa Prado - 10/0039880

1 Introdução

Os robôs estão cada vez mais autônomos, e quando se trata de exploração de um ambiente, uma importante tarefa que ele deve estar apto a realizar é a prevenção de obstáculos em seu caminho. Essa tarefa é dinâmica, ou seja, não é necessário ter um conhecimento prévio do ambiente em que o robô se encontra, contando apenas com algoritmos que trabalham com dados recebidos pelos sensores. Tipicamente, o desvio de obstáculos recebe instruções de outro módulo (geralmente o planejador de rota) que indica qual direção ele deve seguir, e assim, decide o movimento que o motor deve realizar baseado nos dados que os sensores fornecem, realizando, portanto, uma interface entre o módulo de decisão de caminho e o motor do robô. Atualmente, existem diversos algoritmos que propõem a resolução do desvio de obstáculos. O utilizado neste projeto foi o Vector Field Histogram(VFH).

2 Metodologia

3 Condições de Funcionamento

4 Algoritmos Implementados

O algoritmo responsável por efetuar o desvio de obstáculos recebe como entrada um vetor indicando a direção e a velocidade a serem aplicadas ao robô e as leituras do laser. Ele retorna um novo vetor representando a direção e velocidade calculadas de modo que o robô tenda a seguir na direção desejada, desviando de eventuais obstáculos. O vetor de direção é representado por uma mensagem do tipo `geometry_msgs/Twist`. As leituras do laser são representadas por mensagens do tipo `sensor_msgs/LaserScan`. Um diagrama contendo os nós e tópicos utilizados pelo sistema podem pode ser observado na Figura 1.

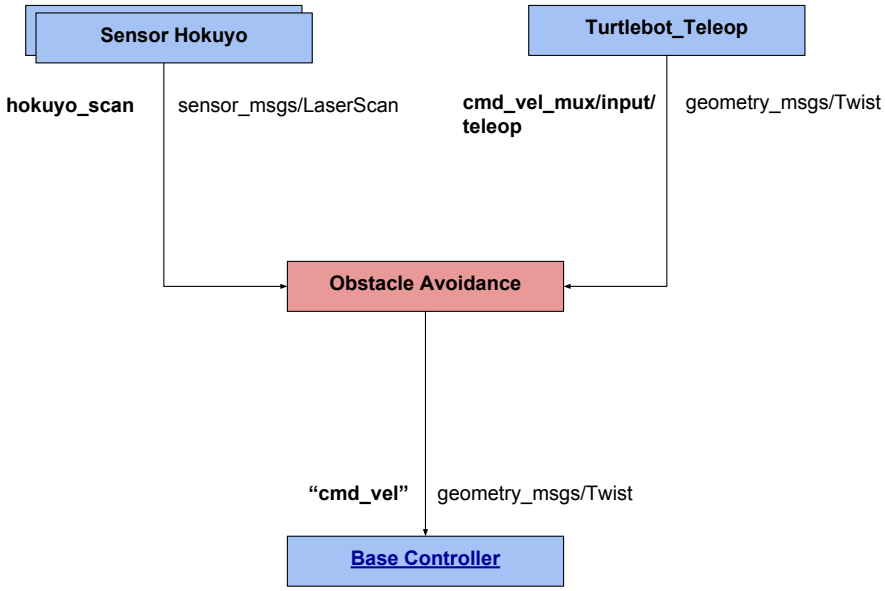

Figura 1: Diagrama de nós do sistema

O tópico `hokuyo_scan` fornece as leituras do laser, e o tópico `cmd_vel_mux/input/` fornece os vetores indicando a direção que o robô deve seguir. O módulo **Obstacle Avoidance** representa o algoritmo de desvio de obstáculos, o qual posta os vetores resultantes no tópico `cmd_vel` do robô.

4.1 Detecção de Obstáculos

Foi utilizado um algoritmo do tipo “bolha”, o qual detecta a presença de obstáculos que ultrapassem uma determinada distância limite (envoltório da bolha). Esta distância limite visa assegurar que o robô evite colisões. Caso alguma amostra do sensor aponte um valor de distância menor que o limite, é chamado o algoritmo VFH. Como o laser utilizado fornece aproximadamente 728 amostras de distâncias (*ranges*), 40 vezes por segundo, decidiu-se verificar a cada dez amostras a presença de obstáculos, de -90° a $+90^\circ$ em relação ao eixo central do robô (Figura 2). O algoritmo de detecção de obstáculos implementado pode ser observado no Código 1.

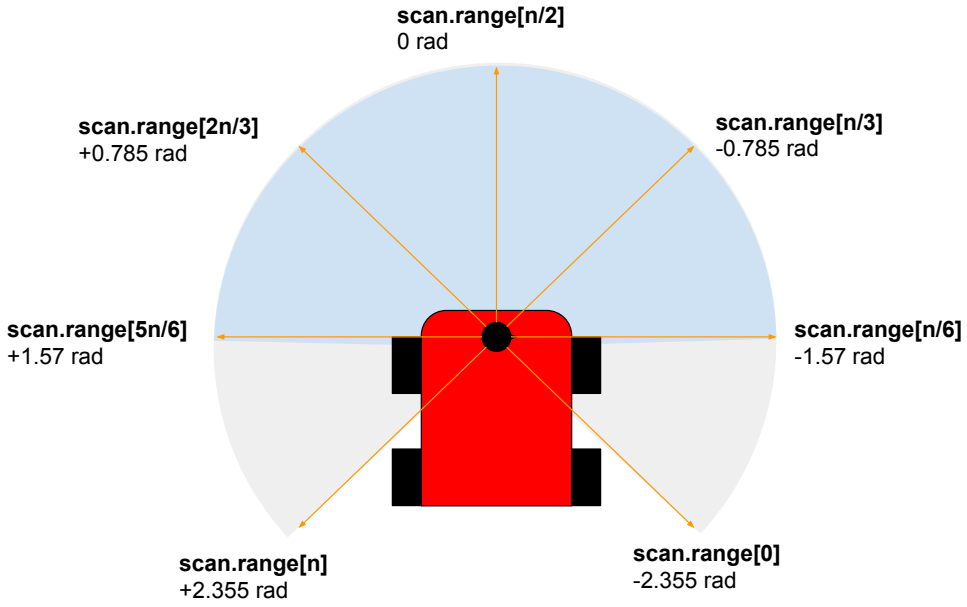

Figura 2: Leitura do sensor laser

Código 1: Algoritmo de Verificação de Obstáculos

```
bool checkForObstacles(sensor_msgs::LaserScan msg, float limite){

    int numero_amostras = (int) floor((msg.angle_max - msg.angle_min) / msg
        .angle_increment);

    //Le as amostras a cada 10 unidades
    for (int i = 0; i < numero_amostras; i+= 10){
        if(i > numero_amostras/6 && i < 5*numero_amostras/6) //90 graus
            if (msg.ranges[i] < limite) return true;
    }
    return false;
}
```

4.2 VFH

Primeiramente, o VFH monta um histograma polar de magnitudes (Figura 3), de de -90° a $+90^{\circ}$ em relação ao eixo central do robô. Em seguida verifica os possíveis vales para onde o robô poderá seguir, retornando o conjunto de ângulos que representam as direções centrais de cada vale (Código 3).

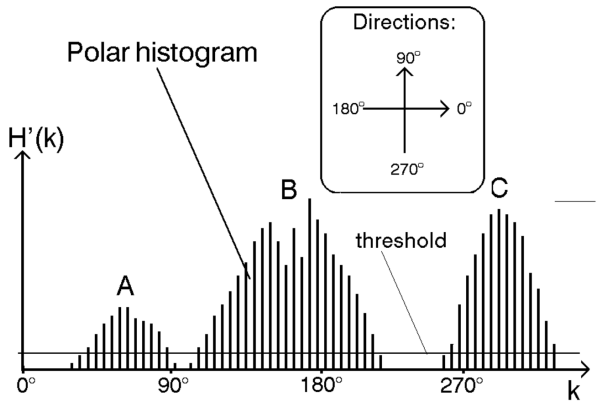

Figura 3: Histograma polar de magnitudes, com um valor empírico de limiar(threshold) de definição de um vale. Fonte: [1]

Código 2: Algoritmo de Verificação de Vales

```
int VFH::getVales(float* vales, sensor_msgs::LaserScan msg){

    int sec_counter = 0; //Contador de setores por vale
    int num_vales = 0; //Contador de setores por vale

    //A cada setor de -angulo_abertura a angulo_abertura, verifica os
        possiveis vales
    for (float alpha = (-1)*angulo_abertura; alpha < angulo_abertura; alpha
         += angulo_setor){

        float c = 1; //Probabilidade de haver um obstaculo no setor
        float dist_setor = getDistanceAverage(alpha, msg, 5); //Range do
            setor atual
        float m = pow(c,2) * (a - b * dist_setor); //Calcula magnitude

        //Verifica possiveis vales
        //Se o valor atual for maior que o limite (ou percorreu todas as
            amostras)
        if (m > limite || alpha+angulo_setor >= angulo_abertura){
            if (sec_counter >= s_max) //Se o vale tiver o numero minimo de
                setores
                vales[num_vales++] = alpha - (sec_counter * angulo_setor)
                    /2; //Calcula angulo central resultante do vale

            //Reseta o contador
            sec_counter = 0;
        }
        else sec_counter++;
    }
    return num_vales;
}
```

Se nenhum vale for encontrado, o robô para e gira em torno do próprio eixo. Caso sejam encontrados um ou mais vales, é escolhido o vale cuja direção seja a mais próxima da direção em que o robô é ordenado a seguir. Durante este processo a velocidade do robô é limitada, de modo

que ele não venha a colidir com o obstáculo do qual ele está desviando.

Código 3: Escolha da direção e velocidade

```
(...)
  //Se nao encontrou nenhum vale
    if (num_vales == 0){
        twist_teleop.linear.x = 0; //Fica parado
        twist_teleop.angular.z = -1; //E girando em torno do proprio eixo
    }

    //Encontrou vale(s)
    else {
        //Escolhe o angulo referente ao vale mais proximo
        twist_teleop.angular.z = nearestAngle(twist_teleop.angular.z, vales
            , num_vales);

        //Controle da velocidade
        float c = 1; //Probabilidade de haver um obstaculo no setor
        float dist_vale = getDistanceAverage(twist_teleop.angular.z, msg,
            5); //Range do vale escolhido
        float m = pow(c,2) * (a - b*dist_vale); //Calcula magnitude

        twist_teleop.linear.x *= (1 - fmin(m, hm)/hm); //Controla a
            velocidade do robo pela magnitude do vale

        //Controla a velocidade pela diferenca angular
        twist_teleop.linear.x = twist_teleop.linear.x *
            (1 - fabs(1.5 * twist_teleop.angular.z)/angulo_abertura) +
                MIN_SPEED;
    }

    //Retorna o vetor resultante
    return twist_teleop;
```

4.3 Leituras do Laser

Para as amostras do sensor, foi utilizado um filtro baseado na média temporal dos valores fornecidos, visando a eliminação de eventuais ruídos na leitura (Código 4).

Código 4: Filtro de amostras do laser

```
void laserCallback(sensor_msgs::LaserScan scan)
{

    laserScanBuffer.push_back(scan);
    if(laserScanBuffer.size() > 40)
    {
        // Erases oldest element
        laserScanBuffer.erase(laserScanBuffer.begin());
    }

    scan_mem = scan;
    //ROS_INFO("%f\t%f", scan.range_min, scan.range_max);
    for(auto laserScan : laserScanBuffer)
    {
```

```
            for(auto range = scan.range_min ; range < scan.range_max ; range++)
            {
                scan_mem.ranges[range] += laserScan.ranges[range];
            }
        }

        for(auto range = scan.range_min ; range < scan.range_max ; range++)
        {
            scan_mem.ranges[range] /= laserScanBuffer.size();
        }

        //Informa inicializacao da scan_mem
        scan_mem_active = 1;

}
```

5 Conclusão

6 Referências

[1] THE VECTOR FIELD HISTOGRAM - FAST OBSTACLE AVOIDANCE FOR MOBILE ROBOTS, J. Borenstein, Member, IEEE and Y. Koren, Senior Member, IEEE The University of Michigan, Ann Arbor Advanced Technology Laboratories http://www-personal.umich.edu/ johannb/Papers/paper16.pdf (Acessado em: 01/dez/2016)