ROBOCUP@HOME

ROBOTITOS

Programming ‘Carry my luggage’

{

< Prueba 1: Carry My Luggage >

< En esta prueba, el robot tendrá que entrar a la arena en busca del árbitro y seguirle hasta cierto punto sin mapear llevando su equipaje>

< se intentó ;(>

}

Introduction;

‘Transcurso de la prueba’

1. El robot empezará en el punto de inicio y tendrá que entrar dentro de la arena.

2. Dentro de la arena, debe dirigirse a la posición del arbitro.

3. El arbitro, mediante su dedo o la voz, indicará que bolsa quiere y el robot tendrá que acercarse a ella.

4. El robot tendrá que pedir que le pongan la bolsa encima y que le indiquen cuando empezar el seguimiento.

5. El robot tendrá que seguir al arbitro por la arena, donde habrá obstáculos.

6. El robot tendrá que seguir al arbitro por fuera de la arena hasta que este se lo indique.
7. El robot tendrá que volver a la arena.

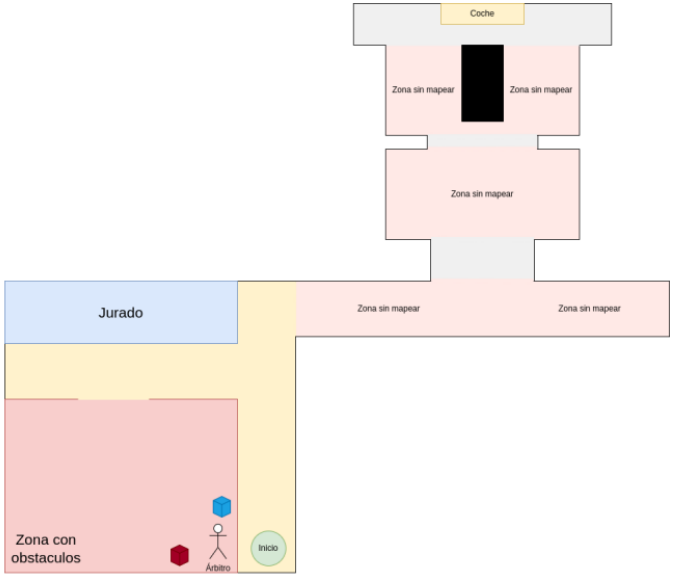
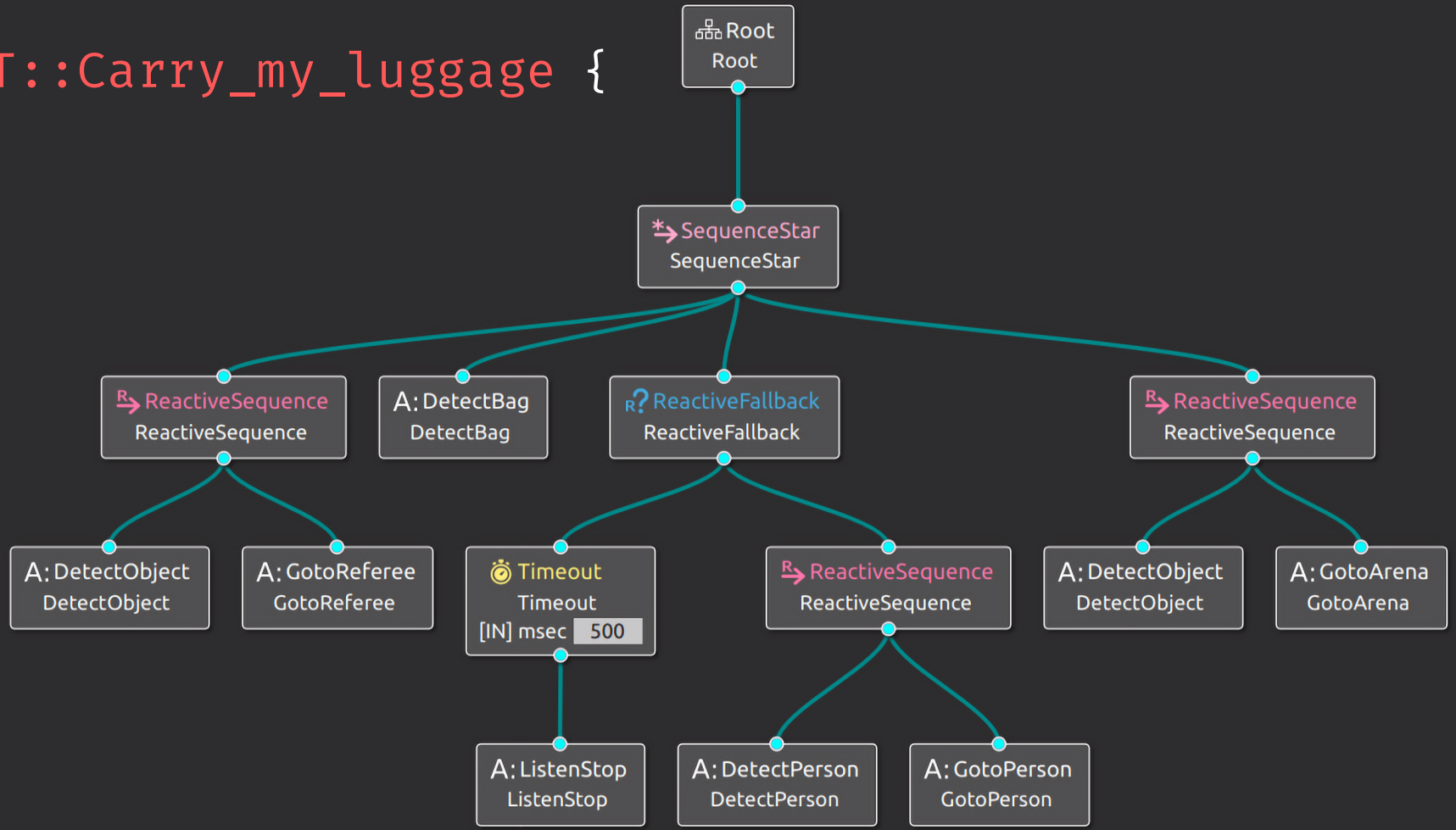


Figure 1: Escenario Carry My Luggage.

BT::Carry_my_luggage {



}

```
Table Of ‘Steps’ {

    Step 01  GotoReferee

        Step 02  DetectBag

            Step 03  FollowPerson

                Step 04  ReturntoArena

}
```

01 {

[GotoReferee]

< Entrar a la Arena y moverse a la localización del arbitro>

< Se envia la posisción y la orientación de donde se encuentra el arbitro a la navegación >

}

02 {

[DetectBag]

< El arbitro, mediante su dedo, indicará que bolsa quiere y el robot se girara viendo la bolsa. El robot tendrá que pedir que le pongan la bolsa encima y que le indiquen cuando empezar el seguimiento >

}

DetectBag{

< Mediante el Bounding box del darknet se calculará que maleta se escogió viendo el centro del cuadrado hacia que lado se movió >

}

```cpp
void DetectBag::DetectBagCallBack(const darknet_ros_msgs::BoundingBoxesConstPtr& boxes) {
  if (fowarder.intent_ == "DetectBag"){
    for (const auto & box : boxes->bounding_boxes) {
      if (box.Class == "person") {
        if (!found_person_){
          found_person_ = true;
          px_init = (box.xmax + box.xmin) / 2;
          py_init = (box.ymax + box.ymin) / 2;
        }
        px = (box.xmax + box.xmin) / 2;
        py = (box.ymax + box.ymin) / 2;

        if ((px - px_init) > 4 && (fowarder.intent_ == "DetectBag")){
          if ((left_counter_ >= 3))
          {
            std::cerr << "Izquierda" << std::endl;
          }
          if (!turning_done)
          {
            left_counter_++;
            right_counter_ = 0;
          }
        }
        else if ((px - px_init < -4) && (fowarder.intent_ == "DetectBag")){
          if (right_counter_ >= 3){
            std::cerr << "Derecha" << std::endl;
          }
          if(!turning_done)
          {
            right_counter_++;
            left_counter_ = 0;
          }
        }
      }
    }
  }
}
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```cpp
BT::NodeStatus DetectBag::tick()
{
  dialogflow_ros_msgs::DialogflowResult result;
  switch (state_) {
      case IDLE:
        if (fowarder.intent_ == "Default Fallback Intent") {
          state_ = LISTEN;
          break;
        }
        if ((right_counter_ >= 3) && ( fowarder.intent_ == "DetectBag")) {
          turn_ts_ = ros::Time::now();
          state_ = TURN;
          break;
        }
        else if ((left_counter_ >= 3) && ( fowarder.intent_ == "DetectBag")) {
          turn_ts_ = ros::Time::now();
          state_ = TURN;
          break;
        }
        else if (fowarder.intent_ == "GO") {
          return BT::NodeStatus::SUCCESS;
        }
        state_ = LISTEN;
        break;
      case TURN:
        if (right_counter_ > 3) {
          if ((ros::Time::now() -turn_ts_).toSec() < 1){
            turning_done = true;
            cmd.angular.z = 0.5;
          }
          else {
            speak_ts_ = ros::Time::now();
            turning_done = true;
            cmd.angular.z = 0;
            fowarder.intent_ = "GO";
            state_ = SPEAK;
          }
        } else if (left_counter_ > 3) {
          if ((ros::Time::now() -turn_ts_).toSec() < 3){
            cmd.angular.z = -0.5;
            turning_done = true;
          }
          else {
            speak_ts_ = ros::Time::now();
            turning_done = true;
            cmd.angular.z = 0;
            fowarder.intent_ = "GO";
            state_ = SPEAK;
          }
        }
        pub_vel_.publish(cmd);
        break;
```

```cpp
      case LISTEN:
        if ( fowarder.intent_ == "DetectBag") {
          state_ = IDLE;
          break;
        }
        speak_ts_ = ros::Time::now();
        cmd.angular.z = 0.0;
        state_ = SPEAK;
        break;
      case SPEAK:
        if (fowarder.intent_ == "INIT") {
          fowarder.speak("Choose one bag");
          fowarder.intent_ = "DetectBag";
        }
        if (fowarder.intent_ == "GO") {
          fowarder.speak("Ok, I follow you now");
          fowarder.intent_ == "DetectBag";
        }
        if ((ros::Time::now() - speak_ts_).toSec() >= 3) {
          state_ = IDLE;
        }
        break;
    }
    return BT::NodeStatus::RUNNING;
}
```

Función tick() del DetectBag

03 {

[FollowPerson]

< El robot tendrá que seguir al arbitro por dentro y fuera de la arena, evitando Obstáculos dinámicos, hasta que éste se lo indique >

}

FollowPerson; {

ListenStop

< Siempre escuchando algún "Stop" para que el robot proceda a volver a la arena >

DetectPerson

< Detecta con Darknet a la persona, obtiene su posición aproximada y publica la tf >

GotoPerson

< Con la tf se obtiene su posición y orientacion y con navegacion se llega a la persona >

}

DetectPerson{

< Publicará una tf en el mapa con la posición de la persona que ha sido detectada mediante la Bounding Box de Darknet >

}

```cpp
void DetectPerson::cloudCB(const sensor_msgs::PointCloud2::ConstPtr& cloud_in)
{
  sensor_msgs::PointCloud2 cloud;
  pcl_ros::transformPointCloud(workingFrameId_, *cloud_in, cloud, tfListener_);

  pcl::PointCloud<pcl::PointXYZRGB>::Ptr pcrgb(new pcl::PointCloud<pcl::PointXYZRGB>);
  pcl::fromROSMsg(cloud, *pcrgb);

  int pixel =py*image_width + px;
  auto point = pcrgb->at(pixel);

  tf2::Stamped<tf2::Transform> transform;
  if (found_person_) {
    if (!std::isnan(point.x) && !std::isnan(point.y))
    {
      transform.setOrigin(tf2::Vector3(point.x, point.y, 0));
      transform.setRotation(tf2::Quaternion(0.0, 0.0, 0.0, 1.0));
      transform.stamp_ = ros::Time::now();
      transform.frame_id_ = workingFrameId_;
      geometry_msgs::TransformStamped object_msg = tf2::toMsg(transform);
      object_msg.child_frame_id = objectFrameId_;
      tfBroadcaster_.sendTransform(object_msg);
    }
  }
}

void DetectPerson::callback_bbx(const darknet_ros_msgs::BoundingBoxesConstPtr& boxes){
  for (const auto & box : boxes->bounding_boxes) {
    if (box.Class == "person") {
      px = (box.xmax + box.xmin) / 2;
      py = (box.ymax + box.ymin) / 2;
      found_person_ = true;
    }
  }
}
```

GotoPerson{

< Se obtiene las posición de la persona y se envia a la navegación >

< La orientación se obtiene del robot >

}

```cpp
BT::NodeStatus GotoPerson::on_tick() {
  move_base_msgs::MoveBaseGoal goal;

  if (counter_++ == 10)
  {
    if (buffer.canTransform("map", "base_footprint", ros::Time(0), ros::Duration(0.1), &error))
    {
      map2bf_msg = buffer.lookupTransform("map", "base_footprint", ros::Time(0));
      tf2::fromMsg(map2bf_msg, map2bf);
    }

    if (buffer.canTransform("base_footprint", "odom", ros::Time(0), ros::Duration(0.1), &error))
    {
      bf2odom_msg = buffer.lookupTransform("base_footprint", "odom", ros::Time(0));
      tf2::fromMsg(bf2odom_msg, bf2odom);
    }

    if (buffer.canTransform("odom", "object/0", ros::Time(0), ros::Duration(0.1), &error))
    {
      odom2obj_msg = buffer.lookupTransform("odom", "object/0", ros::Time(0));
      tf2::fromMsg(odom2obj_msg, odom2obj);
    }

    bf2obj = map2bf * bf2odom * odom2obj;
    goal.target_pose.header.frame_id = "map";
    goal.target_pose.header.stamp = ros::Time::now();
    goal.target_pose.pose.position.x = bf2obj.getOrigin().x();
    goal.target_pose.pose.position.y = bf2obj.getOrigin().y();
    goal.target_pose.pose.position.z = bf2obj.getOrigin().z();
    goal.target_pose.pose.orientation.x = directions.target_pose.pose.orientation.x;
    goal.target_pose.pose.orientation.y = directions.target_pose.pose.orientation.y;
    goal.target_pose.pose.orientation.z = directions.target_pose.pose.orientation.z;
    goal.target_pose.pose.orientation.w = directions.target_pose.pose.orientation.w;

    set_goal(goal);
    counter_ = 0;
  }
}
```

04 {

[ReturntoArena]

< El robot tendrá que volver a la arena >
< Se envia la posisción de la arena a la navegación >

}

Programming ‘Find My Mates’

{

< Prueba 2: Find My Mates >

< En esta prueba, el robot tendrá que entrar a la arena en busca de sus compañeros y notificar al arbitro el nombre de estos e información extra de ellos >

< El fantasma nos lo arruinó ;(>

}

Introduction;

‘Transcurso de la prueba’

1. El robot empezará en el punto de inicio y tendrá que entrar dentro de la arena.

2. Dentro de la arena, tendrá que localizar a la primera persona.

3. Una vez localizada, tendrá que acercarse a ella.

4. El robot preguntará el nombre y obtendrá el color de camiseta y el objeto que lleva la persona sentada.

5. Después, el robot tendrá que ir al árbitro a reportar la información obtenida o seguir buscando personas y reportar todas al final.

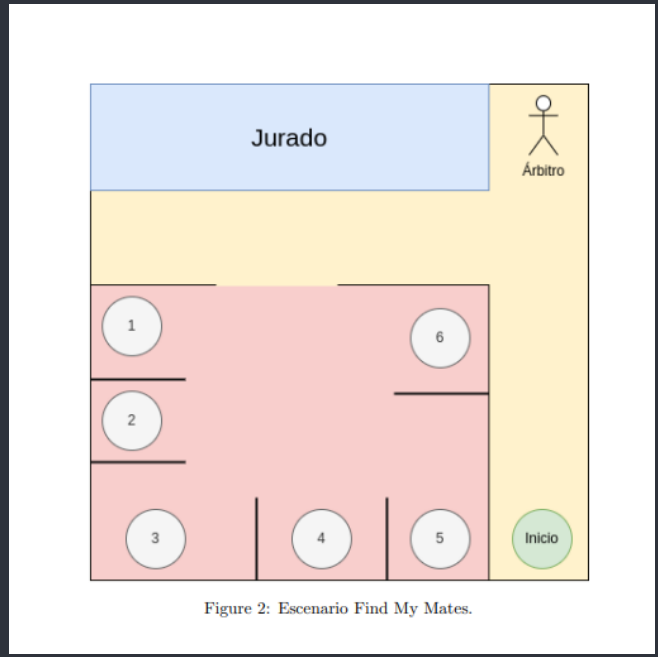
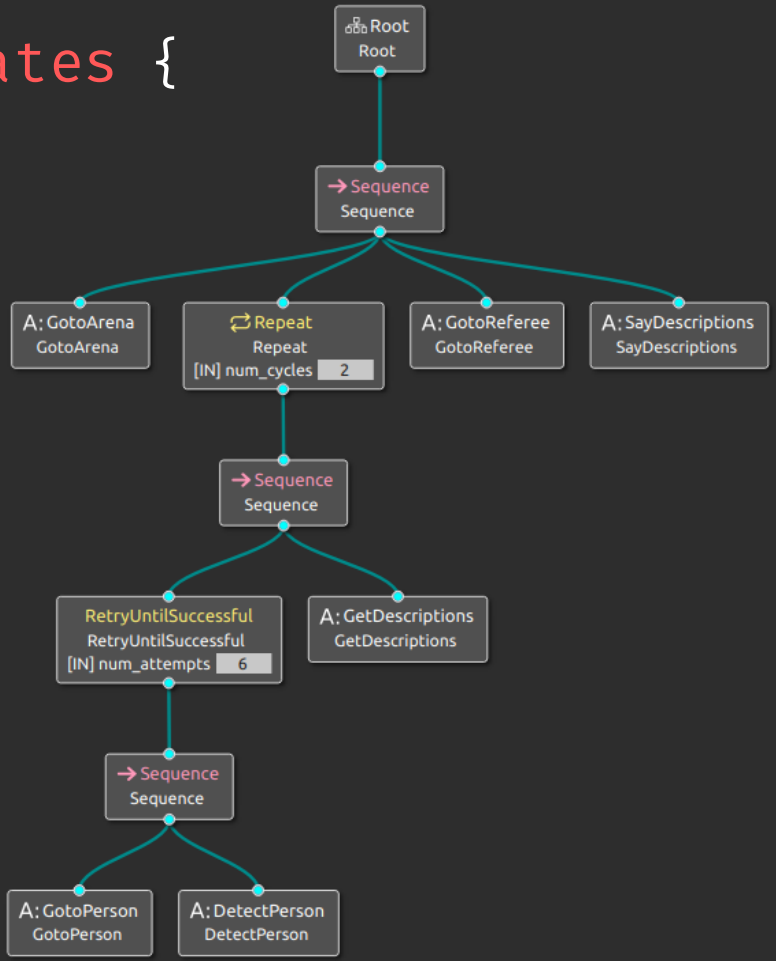


Figure 2: Escenario Find My Mates.

BT::Find_my_mates {



}

Table Of ‘Steps’ {

01 FindPerson

< Entra a la Arena en busca de una persona >

02 GetDescriptions

< Obtiene las descripciones de la persona >

03 SayDescriptions

< Se dirige al árbitro y le reporta las descripciones >

}

01 {

[FindPerson]

< El robot entrará a la arena buscando en cada posición a alguna persona >

}

02 {

[GetDescriptions]

< El robot, mediante diálogo y Darknet, obtendrá la descripción de la persona y el objeto que esta porta >

}

03 {

[SayDescriptions]

< El robot tendrá que ir al árbitro a reportar la información obtenida >

}

Muchas Gracias {

//Preguntas

ROBOTITOS:

< /1 > Oriana Acosta
< /2 > Jaime Avilleira
< /3 > Juan Fernandez
< /4 > Francisco Gomez

}

Asi quedó nuestro cerebro con tantos errores