

# PROYECTO FINAL: ROBOT ACOMODADOR



Por:  
Mario Casero  
Rebeca Castilla  
Henar Contreras  
Lorea Vera

# RESUMEN

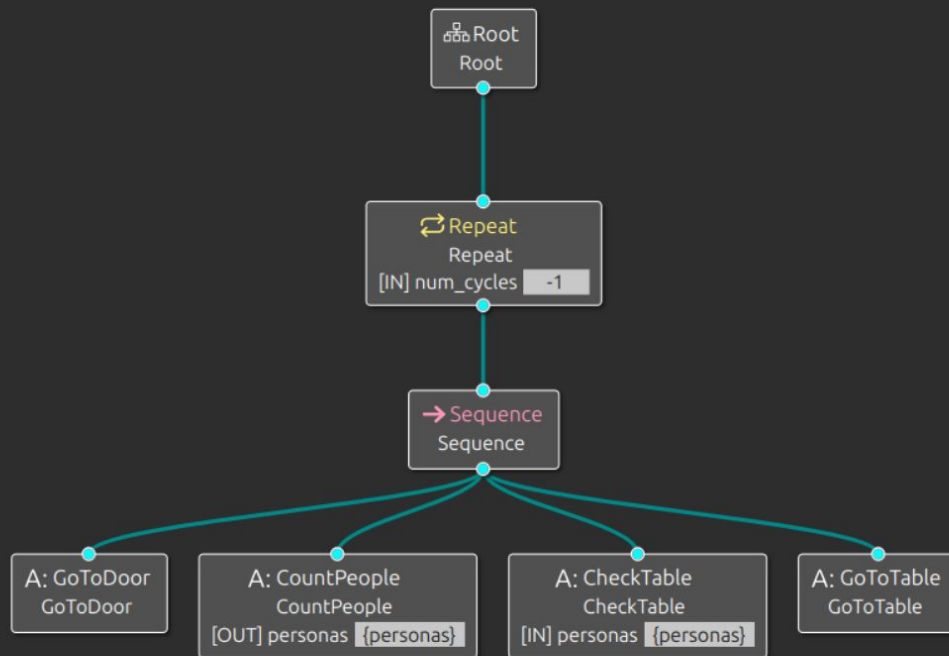
La idea principal es la implementación de un acomodador de un restaurante.

El kobuki irá desde su posición inicial a un punto donde se encontrará una cantidad  $X$  de personas, mediante los hri, el kobuki preguntará cuántos son, y dependiendo de la cantidad les llevará a una mesa u otra, para luego volver al punto donde se encuentran los clientes. Si la cantidad de personas es superior a 6 o no hay mesas libres, el kobuki dirá que no hay mesas disponibles.



# BT

★ MainTree



# IMPLEMENTACIÓN 1: GoToDoor

El kobuki irá desde su posición inicial hasta el grupo de personas.

```
BT::NodeStatus GoToDoor::tick()
{
    // Objetivo fijo
    double x = -2.479227443090446;
    double y = -2.144102269394439;
    double theta = 0.0;

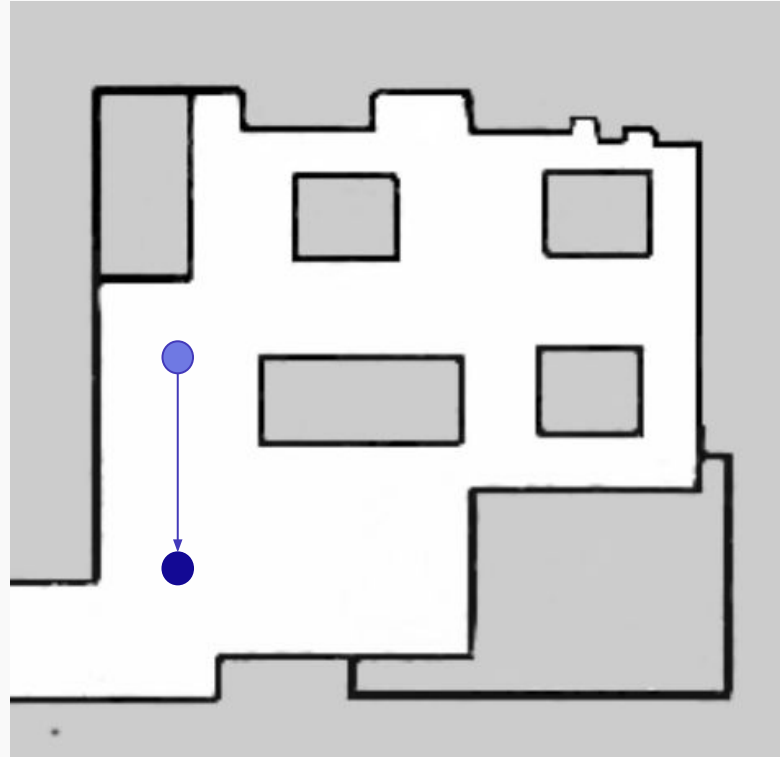
    geometry_msgs::msg::PoseStamped goal;
    goal.header.frame_id = "map";
    goal.header.stamp = node_->now();
    goal.pose.position.x = x;
    goal.pose.position.y = y;
    goal.pose.position.z = 0.0;

    tf2::Quaternion q;
    q.setRPY(0, 0, theta);
    goal.pose.orientation = tf2::toMsg(q);

    NavigateToPose::Goal nav_goal;
    nav_goal.pose = goal;

    auto start_time = node_->now();

    auto send_goal_future = action_client->async_send_goal(nav_goal);
```



```
auto goal_handle = send_goal_future.get();
if (!goal_handle) {
    RCLCPP_ERROR(node->get_logger(), "Goal was rejected by server");
    return BT::NodeStatus::FAILURE;
}

auto result_future = action_client->async_get_result(goal_handle);
if (rclcpp::spin_until_future_complete(node_, result_future) != rclcpp::FutureReturnCode::SUCCESS) {
    RCLCPP_ERROR(node->get_logger(), "Failed while waiting for result");
    return BT::NodeStatus::FAILURE;
}

auto result = result_future.get();
auto end_time = node->now();

rclcpp::Duration duration = end_time - start_time;

if (result.code == rclcpp_action::ResultCode::SUCCEEDED) {
    RCLCPP_INFO(node->get_logger(), "Navigation succeeded in %.2f seconds", duration.seconds);

    // Publica cmd_vel en cero para detener el robot
    geometry_msgs::msg::Twist stop_vel;
    stop_vel.linear.x = 0.0;
    stop_vel.angular.z = 0.0;
    cmd_vel_pub->publish(stop_vel);

    return BT::NodeStatus::SUCCESS;
} else {
    RCLCPP_ERROR(node->get_logger(), "Navigation failed");
    return BT::NodeStatus::FAILURE;
}
}
```

# IMPLEMENTACIÓN 2: CountPeople

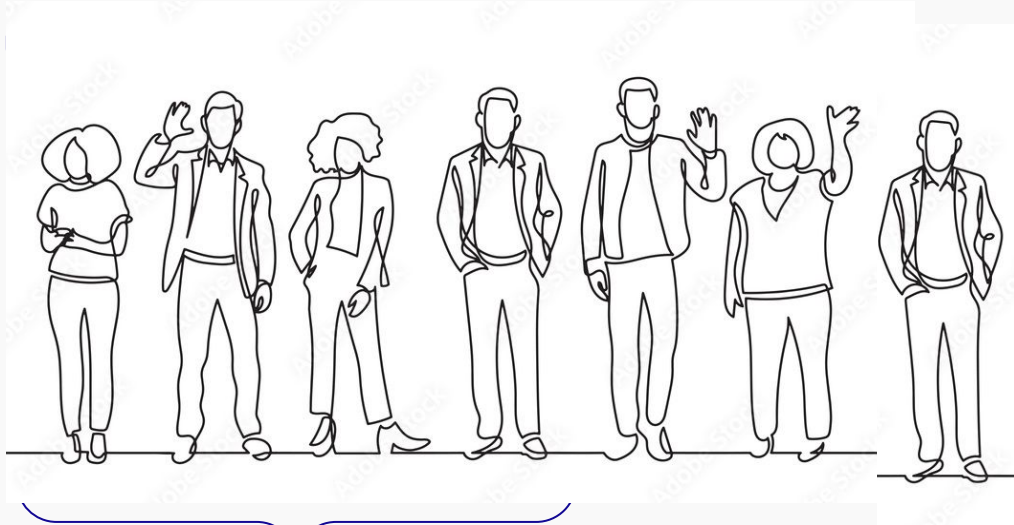
El kobuki preguntará por el n de personas.

```
bool CountPeople::contarPersonas()
{
    int personas = 0;
    hablarFestival("Por favor, introduzca para cuántos será la mesa");
    std::cout << "¿Mesa para cuántos?\n";
    std::cin >> personas;

    // Validamos que el número ingresado sea válido
    if (std::cin.fail() || personas <= 0) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        hablarFestival("Número inválido");
        std::cerr << "Número inválido.\n";
        return false; // Si es inválido, devolvemos false
    }

    // Guardamos el número de personas en la blackboard
    if (!setOutput("personas", personas)) {
        std::cerr << "No se pudo guardar el número de personas en la blackboard.\n";
        return false;
    }

    hablarFestival("Buscando su mesa, por favor espere");
    std::cout << "Buscando mesa para " << personas << std::endl;
    return true;
}
```



SMALL TABLE

BIG TABLE

NO TABLES AVAILABLE

```

void CountPeople::crearMesas()
{
    Mesa big{6, false}; // BIG: tamaño 6
    Mesa small{4, false}; // SMALL: tamaño 4

    // Guardamos las mesas en la blackboard
    auto bb = config().blackboard;
    bb->set("mesa_big", big);
    bb->set("mesa_small", small);

    // Imprimimos el estado de las mesas
    std::cout << "Mesas creadas y guardadas en la blackboard:";
    std::cout << " - mesa_big: " << big << "\n";
    std::cout << " - mesa_small: " << small << "\n";
}

} // namespace controlper

#include "behaviortree_cpp_v3/bt_factory.h"
BT_REGISTER_NODES(factory)
{
    factory.registerNodeType<controlper::CountPeople>("CountPeople");
}

/
{
    Mesa big{6, false};
    Mesa small{4, false};

    auto bb = config().blackboard;
    bb->set("mesa_big", big);
    bb->set("mesa_small", small);

    std::cout << "Mesas creadas y guardadas en la blackboard:\n";
    std::cout << " - mesa_big: " << big << "\n";
    std::cout << " - mesa_small: " << small << "\n";
}

```

```

CountPeople::CountPeople(const std::string& name, const BT::NodeConfiguration& config
    : BT::SyncActionNode(name, config) {}

// Define los puertos proporcionados por este nodo
BT::PortsList CountPeople::providedPorts() // puerto de salida de personas
{
    return { BT::OutputPort<int>("personas") };
}

BT::NodeStatus CountPeople::tick()
{
    std::cout << "CountPeople ejecutandose" << std::endl;

    if (!contarPersonas()) {
        return BT::NodeStatus::FAILURE;
    }

    // Si contar personas tiene éxito, creamos las mesas
    crearMesas();

    return BT::NodeStatus::SUCCESS;
}

```

Con personas se usa output port  
Con las mesas se usa un set



## IMPLEMENTACIÓN 3: CheckTable

El kobuki asignará una mesa en función de si está libre y del número de personas

```

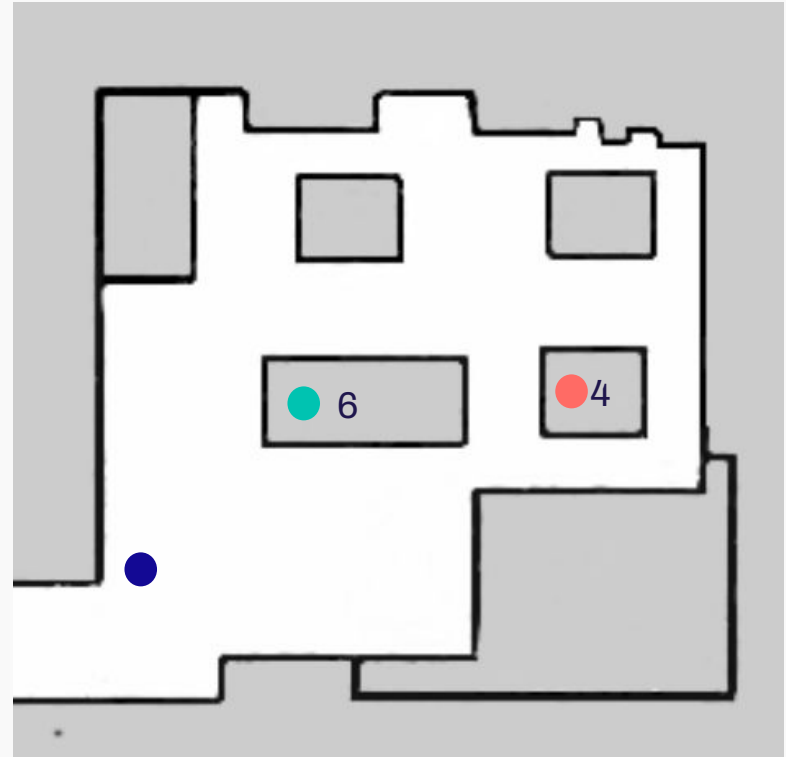
if (!bb->get("mesa big" |> |bb->get("mesa_small", small)) {
    std::cerr << "No se pudieron obtener las mesas desde la blackboard." << std::endl;
    return BT::NodeStatus::FAILURE;
}

if (personas > 6) {
    hablarFestival(texto: "Ahora mismo no se encuentra una mesa disponible, lo sentimos");
    std::cout << "Mesa no disponible para " << personas << " personas." << std::endl;
    return BT::NodeStatus::FAILURE;
}

else if (personas >= 4) {
    if (big.llena) {
        hablarFestival(texto: "Ahora mismo no se encuentra una mesa disponible, lo sentimos");
        std::cout << "No hay mesas disponibles (mesa BIG ya ocupada)." << std::endl;
        return BT::NodeStatus::FAILURE;
    }
    hablarFestival(texto: "Asignando mesa");
    std::cout << "Asignando a mesa BIG..." << std::endl;
    big.llena = true;
    bb->set("mesa big", big);
    bb->set("destino", std::string(s: "BIG"));
    std::cout << "Estado actualizado: " << big << std::endl;
}

else {
    if (small.llena) {
        hablarFestival(texto: "Ahora mismo no se encuentra una mesa disponible, lo sentimos");
        std::cout << "No hay mesas disponibles (mesa SMALL ya ocupada)." << std::endl;
        return BT::NodeStatus::FAILURE;
    }
    hablarFestival(texto: "Asignando mesa");
    std::cout << "Asignando a mesa SMALL..." << std::endl;
    small.llena = true;
    bb->set("mesa_small", small);
    bb->set("destino", std::string(s: "SMALL"));
    std::cout << "Estado actualizado: " << small << std::endl;
}
}

```





# IMPLEMENTACIÓN 4: GoToTable

El kobuki llevará a las personas a su mesa correspondiente y volverá a mirar si han venido más

```
std::cout << "GoToTable ejecutandose" << std::endl;
// Dos objetivos, mesa de seis personas o mesa de cuatro
/* coordenadas mesa BIG (6>n) :
x: -2.6915394668397883
y: -2.809132875211364
z: 0.0

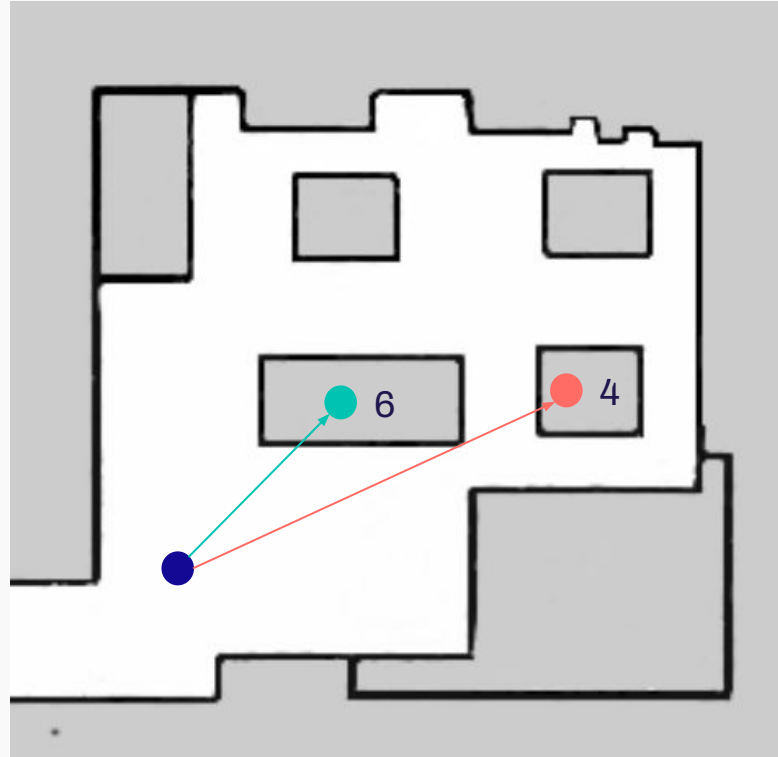
coordenadas mesa SMALL (4>n) :
x: -0.49837595256644057
y: -3.049161757190125
z: 0.0
*/

auto bb = config().blackboard;
std::string destino;

if (!bb->get("destino", destino)) {
    RCLCPP_ERROR(node->get_logger(), "Could not find 'destino' on the blackboard.\n");
    return BT::NodeStatus::FAILURE;
}

double x = 0.0;
double y = 0.0;
double theta = 0.0;

if (destino == "BIG") {
    x = -2.5;
    y = -2.5;
    theta = 0.0;
    RCLCPP_INFO(node->get_logger(), "On path to BIG desk.\n");
} else if (destino == "SMALL") {
    x = 0.0;
    y = -2.5;
    theta = 0.0;
    RCLCPP_INFO(node->get_logger(), "On path to SMALL desk.\n");
} else {
    RCLCPP_ERROR(node->get_logger(), "Unwanted value on variable destino: '%s'\n", destino.c_str());
    return BT::NodeStatus::FAILURE;
}
```



```

geometry_msgs::msg::PoseStamped goal;
goal.header.frame_id = "map";
goal.header.stamp = node_->now();
goal.pose.position.x = x;
goal.pose.position.y = y;
goal.pose.position.z = 0.0;

tf2::Quaternion q;
q.setRPY(0, 0, theta);
goal.pose.orientation = tf2::toMsg(q);

NavigateToPose::Goal nav_goal;
nav_goal.pose = goal;

auto start_time = node_->now(); // Marca de tiempo inicial

auto send_goal_future = action_client->async_send_goal(nav_goal);
if (rclcpp::spin_until_future_complete(node_, send_goal_future) != rclcpp::FutureReturnCode::SUCCESS) {
    RCLCPP_ERROR(node_>get_logger(), "Failed to send goal\n");
    return BT::NodeStatus::FAILURE;
}

auto goal_handle = send_goal_future.get();
if (!goal_handle) {
    RCLCPP_ERROR(node_>get_logger(), "Goal was rejected by server\n");
    return BT::NodeStatus::FAILURE;
}

auto result_future = action_client->async_get_result(goal_handle);
if (rclcpp::spin_until_future_complete(node_, result_future) != rclcpp::FutureReturnCode::SUCCESS) {
    RCLCPP_ERROR(node_>get_logger(), "Failed while waiting for result\n");
    return BT::NodeStatus::FAILURE;
}

auto result = result_future.get();
auto end_time = node_->now(); // Marca de tiempo final

rclcpp::Duration duration = end_time - start_time;

```

```

rclcpp::Duration duration = end_time - start_time;

if (result.code == rclcpp_action::ResultCode::SUCCEEDED) {
    RCLCPP_INFO(node_>get_logger(), "Navigation succeeded in %.2f seconds\n", duration.seconds());
    return BT::NodeStatus::SUCCESS;
} else {
    RCLCPP_ERROR(node_>get_logger(), "Navigation failed\n");
    return BT::NodeStatus::FAILURE;
}
}

```

Apunte: a veces el kobuki se raya y no se ubica bien a la vuelta. Si esto ocurre, el proceso hace halt y para.

Si el kobuki consigue ubicarse bien, vuelve a su posición inicial (con las personas) y las lleva a la mesa en bucle hasta que se decide parar el programa.