Millitec Food Project: RoboChart, RoboSim, and PModel

November 6, 2019

1 RoboChart

detectionBehaviourStm

This state machine imports all the interfaces defined in the platform and controller. It also defines $stmConstI$ and $stmVarI$ interfaces.

stmConstI

HOME_X: The x-coordinate of the home position of the arm.
HOME_Y: The y-coordinate of the home position of the arm.
DELAY_DISCARD: The time needed to calculate whether to discard the first piece of bread or not.
WAIT_PICK and WAIT_DROP: The time needed to pick or drop a piece of bread. In the implementation of the model, we assume that the vertical speed of the arm is fast and we set these values to zero.

intercetOp(x:real, y:real)

This operation imports the *calc_estimate* package and it highly uses the functions defined in that package. It requires $platformConstI$ and $platform OpI$ interfaces. It also defines $interceptOpConstI$ and $interceptOpVarI$ interfaces.

interceptOpConstI

In interceptOp operation, *epsilon_theta* and *epsilon_radius* are used to check whether the distance between the position of arm and the position of bread is less than or equal to epsison or not.

interceptOpVarI

All of the following variabless are used in pid calculations:

t_x: The target x-ccordinate.

t_y: The target y-coordinate.

c_theta: The platform sensor variable that detects current theta of the arm.

c_radius: The platform sensor variable that detects current radius of the arm.

i_theta: Integration error for theta.

i_radius: Integration error for radius.

e_theta: The current error for theta.

e_radius: The current error for radius.

p_e_theta: The previous error for theta.

p_e_radius: The previous error for radius.

d_theta: Derivative error for theta.

e_radius: Derivative error for radius.

t_theta: The target *theta*-ccordinate.

t_radius: The target *radius*-coordinate.

v_theta and *v_radius*: The platform actuator variables to control the velocity of arm in the x-y plane.

Initially, the target (x, y) coordinate of the bread is calculated using the functions *get_t_x* and *get_t_y*, and the previous, derivative, and integral errors are set to zero. Then the target polar coordinate of bread is calculated using the functions *calc_theta* and *calc_radius*. Then the previous errors for theta and radius are calculated using the function *calc_p_e_theta* and *calc_p_e_radius*.

In the state step, the current, derivative, and integration errors for theta and radius are calculated using the corresponding functions. Then the velocity for theta and radius of arm is calculated. These velocities are the actual actuators that move the arm toward the target position using the platform operations *set_v_theta* and *set_v_radius*.

In the joint, we check if the distance between the position of the arm and the position of the bread is less than *epsilon_theta* and *epsilon_radius* or not. If it is less than these values, the velocity of arm is set to zero using the platform operations *set_v_theta* and *set_v_radius*. If not, we need to calculate the new position of the bread because it is moving with the belt. Then, we repeat the state step.

pidOp(x:real, y:real)

This operation is similar to interceptOp. The only difference is that the target position does not change. In interceptOp, the target position is the position of the bread and we know that the bread is moving on the belt. In pidOp, the target position is fixed. It is the home position of the arm.

calc_estimate package

f_x *calc_theta*$(t_x : real, t_y : real) : real$

This function receives the (x, y) coordinate of the target position where the arm is expected to go. It returns θ of the corresponding polar coordinate (r, θ).

f_x *calc_radius*$(t_x : real, t_y : real) : real$

This function receives the (x, y) coordinate of the target position where the arm is expected to go. It returns the *radius* of the corresponding polar coordinate (r, θ).

f_x *update_t_y*$(t\_y : real, speed : real, step\_size : real) : real$

The belt moves. Thus, the target position (the position of the bread) that is sent to interceptOp operation needs to be updated. The x coordinate of the target position will not change but the y coordinate needs to be updated. This is calculated using the *speed* of belt and *step_size*.

f_x *calc_v_radius*$(e\_radius : real, i\_radius : real, d\_radius : real) : real$

This function calculates *v_radius* based on *e_radius* (current error), *i_radius* (integration error), and *d_radius* (derivative error). To do so, it uses the following PID constants for radius: $Pr = 5.2, Ir = 0.2, Dr = 0.4$. *v_radius* is a platform actuator variable for moving the arm in (x,y) plane.

f_x *calc_v_theta*$(e\_theta : real, i\_theta : real, d\_theta : real) : real$

This function calculates v_theta based on e_theta (current error), i_theta (integration error), and d_theta (derivative error). To do so, it uses the following PID constants for theta: $Pt = 6.2, It = 0.2, Dt = 0.4$. v_radius is a platform actuator variable for moving the arm in (x,y) plane.

The rest of functions are defined similarly.

RoboSim

The RoboSim diagrams are similar to RoboChart diagrams. But, they have two extra notions: cycle and exec. Each cycle has three phases: reading sensors, writing to actuators, and executing the operation (denoted by exec), where in fact we wait for the robot to perform the corresponding output operation. For every state of detectionBehaviourStm that shows an operation there are two states in simDetectionBehaviourStm. As an example, for the state LOWER_ON_FIRST_SHAPE, we have two states in simDetectionBehaviourStm: S_LOWER_ON_FIRST_SHAPE (S stands for start)and D_LOWER_ON_FIRST_SHAPE (D stands for during), where in the latter state there is an exec transition that lasts for WAIT_LOWER1 time units so that the arm can actually have enough time to perform lowering on first shape. Note that the events defined in interfaces lCameraI, rCameraI, and FeedbackI are all input of the model and platform operations defined in platformOpI are outputs.

PModel

A PModel (Physical model) shows the physical model of a robot. The PModel of delta_robot consists of some links, bodies, and joints. Any of them may have a position and orientiation with respct to its parent. For example, radius is a joint of type prismatic and forarm is a link. The position of radius is (0.14, 0.0, 0.97) with respect to its parent forarm. The position and orientation for link gripper is not defined. This means that it has the same position and orietation as its parent which is delta_robot. All platform operations and events must be mapped to the corresponding elements in the pmodel through platform mapping.

There is a joint, called height. This joint is of type prismatic. The parent and child of height are links arm and gripper, respectively. Height is reponsible to move the arm vertically. So, it is actually related to the operations lower and lift of the RoboSim model. The lower operation has

a parameter called dist which shows the distance that arm needs to lower down. The local variable $pj2$ of the joint height is set to *0.235-dist* in operation mapping for the operation lower. This is the value of z-coordinate of the arm position after it lowers down. $pj2$ is set to *0.235* when the arms lifts.

There is another joint, called radius. This joint is also of type prismatic. The parent and child of radius are links forarm and arm, repectively. This joint is related to moving the arm in x-y plane. The platform operation *set_v_radius* has a parameter v_r. The local variable $pj1$ of the joint radius is set to the value of v_r whenever the platform operation *set_v_radius* is called in softwate operatoins pidOp and interceptOp.

Finally, there is a joint called theta. This joint is of type revolute. The parent and child of theta are the links base and forarm, respectively. This joint is related to moving the arm in x-y plane. The platform operation *set_v_theta* has a parameter v_t. The local variable rj of the joint radius is set to the value of v_t whenever the platform operation *set_v_theta* is called in softwate operatoins pidOp and interceptOp.

For input event rShapeX, we have an input event mapping: rShapeX.x. It occurs when Camera2.voltage>3.0 and x=Camera2.x.In other words, this event occurs if Camera2 detects a piece of bread and saves the x-coordinate of that bread in variable x. The rest of the events are defined in a similar way.