Universal Robots URCap example picking program

Loading the program

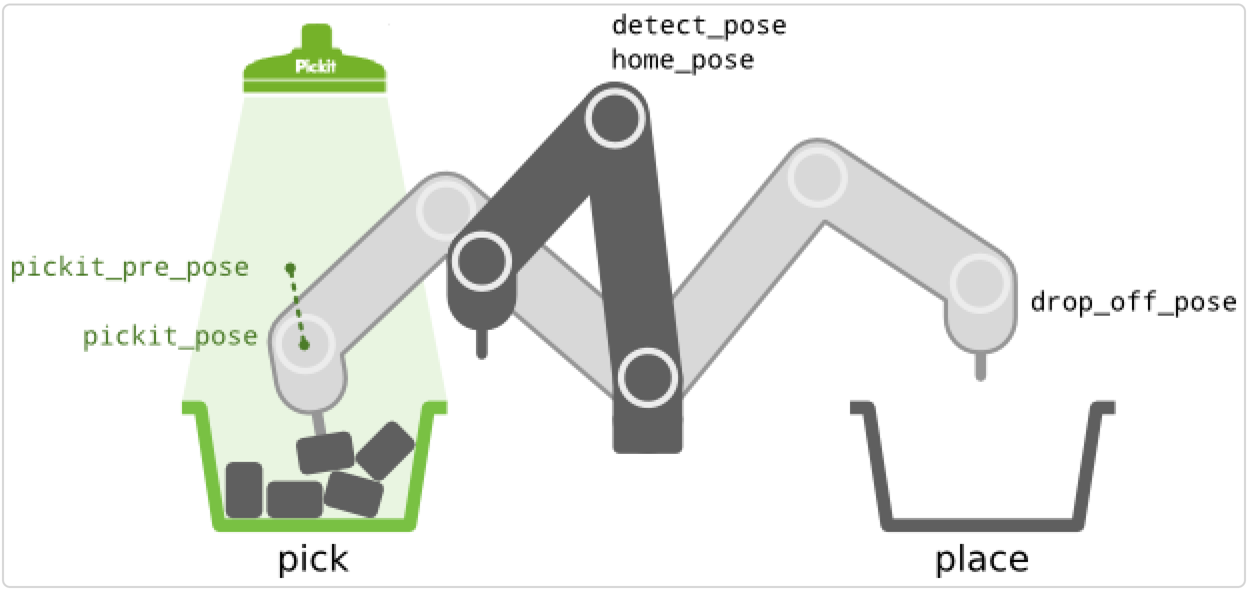
This example program requires the **Pick-it URCap plugin** to be installed in your robot. For installation instructions of both the URCap plugin and the example programs please refer to the Getting started with the Pick-it URCap (//support.pickit3d.com/article/75-getting-started-with-the-pick-it-urcap) article.

In the initialization screen, select `Program robot` and then `Load program`. Navigate to the `pickit_samples/simple_picking` folder and open the `simple_picking.urp` program. The program is then loaded.

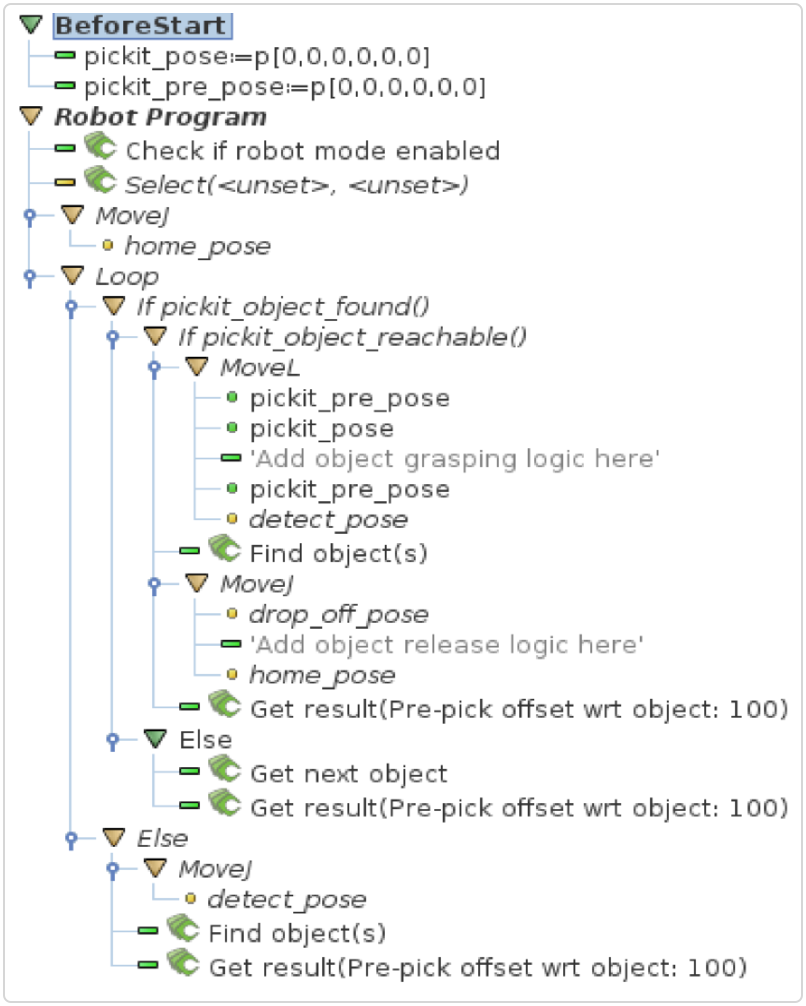
The program explained

The program implements a simple task where Pick-it is used to continuously pick objects from a detection region and place them in a specified location. No assumptions are made on how objects are laid out in the detection region, so the program is a good starting point to build a wide range of applications. Objects can be stacked randomly, or in a pattern, touching or not.

The program assumes a fixed camera mount for simplicity. Please refer to the robot mounted camera (#robot_mounted_camera) advanced topic to learn how to adapt the example to robot mounted camera setups.



The complete program



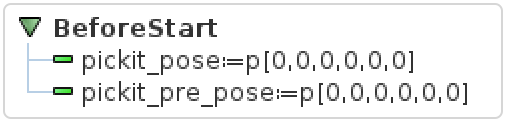
```
BeforeStart
  pickit_pose:=p[0,0,0,0,0,0]
  pickit_pre_pose:=p[0,0,0,0,0,0]
Robot Program
  Check if robot mode enabled
  Select(<unset>, <unset>)
  MoveJ
    home_pose
  Loop
    If pickit_object_found()
      If pickit_object_reachable()
        MoveL
          pickit_pre_pose
          pickit_pose
          'Add object grasping logic here'
          pickit_pre_pose
          detect_pose
        Find object(s)
        MoveJ
          drop_off_pose
          'Add object release logic here'
          home_pose
        Get result(Pre-pick offset wrt object: 100)
      Else
        Get next object
        Get result(Pre-pick offset wrt object: 100)
    Else
      MoveJ
        detect_pose
      Find object(s)
      Get result(Pre-pick offset wrt object: 100)
```

The program is composed by standard URScript commands and commands specific to the Pick-it URCap plugin, which are identified by the green gripper symbol. A complete description of available commands can be found in The Pick-it URCap interface (//support.pickit3d.com/article/80-the-pick-it-urcap-interface) article.

> Commands marked in yellow have uninitialized parameters that must be set by the user before running the program. This is the case for the Pick-it `Select` command, which is explained further down in the document; and for the fixed waypoints `home_pose`, `detect_pose` and `drop_off_pose`, which should be set in accordance to the robot's environment. In particular, we assume a collision-free transition between adjacent waypoints in the program, e.g. between `drop_off_pose` and `home_pose`. We also assume that in `detect_pose` the robot does not occlude the Pick-it camera field of view.

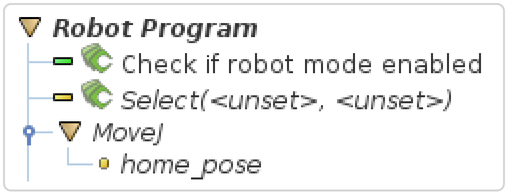
> It's important to have a correctly specified robot TCP, as all `MoveJ` and `MoveL` commands are specified with respect to the **active TCP**. The TCP should be such that when the robot reaches `pickit_pose`, it can establish a successful grasp.

Program breakdown



```
BeforeStart
  pickit_pose:=p[0,0,0,0,0,0]
  pickit_pre_pose:=p[0,0,0,0,0,0]
```

The Pick-it URCap plugin requires the existence of two global variables named `pickit_pose` and `pickit_pre_pose`. The value of these variables is set by the Pick-it plugin after a successful detection.



```
Robot Program
  Check if robot mode enabled
  Select(<unset>, <unset>)
  MoveJ
    home_pose
```

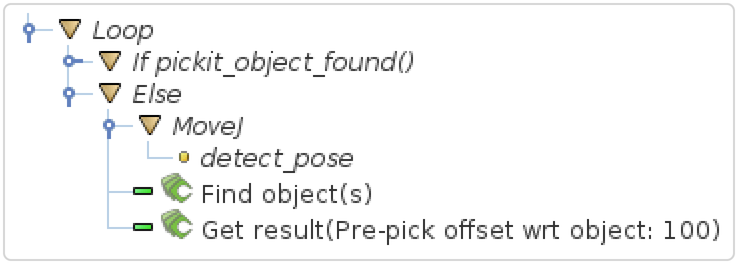
The first commands in the program are run only once. Firstly, the condition that Pick-it is in Robot mode is verified using the `Check if robot mode enabled` command. Next, the `Select` command loads the setup and product configuration to use for object detection. These are initially unset, which is why the command is marked in yellow. This makes it necessary for the user to specify a setup and product configuration from the list of alternatives available on the Pick-it system the robot is connected to.

The robot is then moved to the `home_pose`, from which the detection loop starts. The detection loop is run continuously until the program is stopped, and handles three main cases:

- **No objects detected** → trigger a new Pick-it object detection
- **Object detected and reachable** → perform pick and place sequence and re-trigger object detection
- **Object detected but unreachable** → get another (potentially reachable) object already detected by Pick-it, if any

What follows describes each case in detail. For clarity, code paths related to cases other than the one being described are collapsed.

No object detected



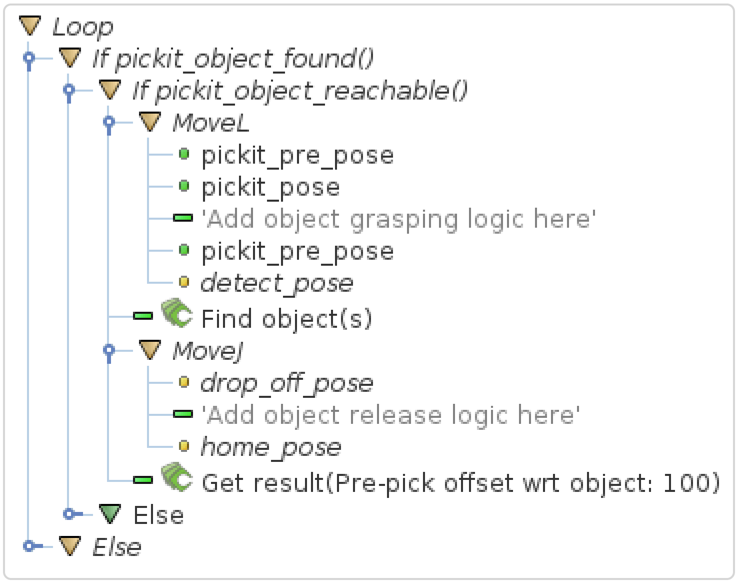
```
Loop
  If pickit_object_found()
  Else
    MoveJ
      detect_pose
    Find object(s)
    Get result(Pre-pick offset wrt object: 100)
```

The detection loop first calls `pickit_object_found()` to check if Pick-it has a detection result available. The first time the loop is entered there are no detection results, as Pick-it has not yet been triggered to detect objects; so the `Else` statement is evaluated.

In the `Else` statement, the robot first moves to `detect_pose`, from which detections are triggered using the `Find object(s)` command. We then wait for Pick-it to reply with detection results using `Get results`. When object detection is successful, this command sets the values of `pickit_pose` and `pickit_pre_pose`: While `pickit_pose` is the actual pose for picking the object, `pickit_pre_pose` is the `pickit_pose` translated by an offset. In this case, this offset is 100mm above the object.

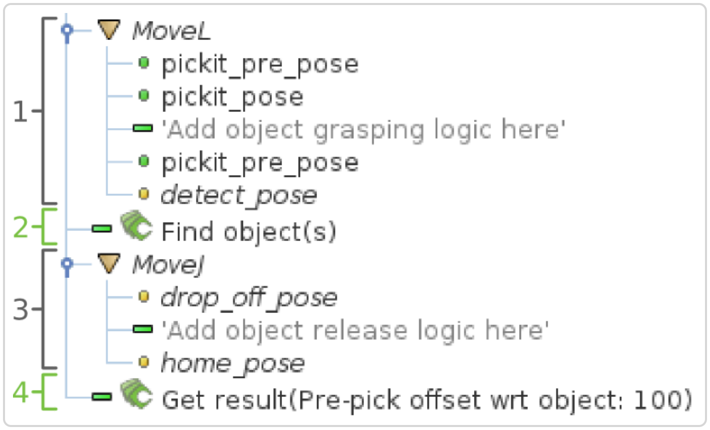
As long as there are no detection results, the robot remains in `detect_pose` continuously looking for objects.

Object detected and reachable



```
Loop
  If pickit_object_found()
    If pickit_object_reachable()
      MoveL
        pickit_pre_pose
        pickit_pose
        'Add object grasping logic here'
        pickit_pre_pose
        detect_pose
      Find object(s)
      MoveJ
        drop_off_pose
        'Add object release logic here'
        home_pose
      Get result(Pre-pick offset wrt object: 100)
    Else
  Else
```

When object detection is successful, `pickit_object_found()` evaluates to `true` and at least one object has been found. We only proceed to pick the object when it is also reachable by the robot. The pick and place sequence is detailed below.



```
1- MoveL
     pickit_pre_pose
     pickit_pose
     'Add object grasping logic here'
     pickit_pre_pose
     detect_pose
2- Find object(s)
3- MoveJ
     drop_off_pose
     'Add object release logic here'
     home_pose
4- Get result(Pre-pick offset wrt object: 100)
```

The sequence consists of four parts:

1. **Pick object** using the pre-pick and pick poses populated by the last call to `Get Result`, and move to the detection pose. Notice linear motions are used.
2. **Trigger object detection**
3. **Place object** in drop-off location and return to the home pose. This takes place in parallel with object detection.
4. **Get object detection results**

Note that `home_pose` can typically be the same as `detect_pose`, but we name the two poses differently to make explicit that in `home_pose` we don't trigger Pick-it detections. As such, `home_pose` is not required to prevent the robot from occluding the camera field of view.

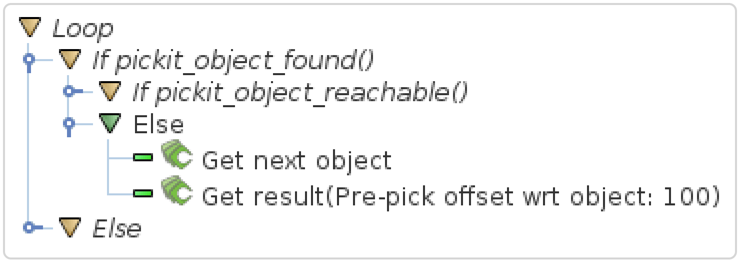
Gripping device

Note that logic specific to the gripping device has been explicitly omitted. You should replace the object grasping and release comments with appropriate commands for controlling your hardware, as running the program as-is will only make the robot point to the object to pick. The gripping device (#gripping device) advanced topic describes an example pick and place sequence for vacuum grippers.

Cycle time optimization

The time it takes for Pick-it to detect objects is application-dependent and can vary from a fraction of a second to multiple seconds. To optimize cycle time, we perform object placing (step 3) in parallel to detection of the next object to pick, such that the wait time when calling `Get Result` is minimal (typically zero). This optimization is relevant in applications with long detection times and tight cycle time constraints. Its benefits are negligible in applications with fast detection times.
The robot mounted camera (https://secure.helpscout.net/docs/583bfcdbc6979106d37373a0/article/5a54e77f2c7d3a194367fbd7/#robot_mounted_camera) advanced topic presents an alternative pick and place sequence where no cycle time optimization is done.

Object detected but unreachable



```
Loop
  If pickit_object_found()
    If pickit_object_reachable()
    Else
      Get next object
      Get result(Pre-pick offset wrt object: 100)
  Else
```

When an object is detected but unreachable, the detection loop checks if there are additional object detections available from Pick-it. A single call to `Find object(s)` might yield the detection of multiple objects, and `Get next object` allows to access the next available object, if any, without the need of triggering a new detection and the time overhead it entails.

Running the program

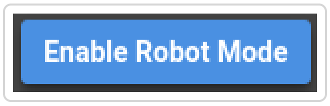
Program execution

When running a program for the first time, it is advised to **set a low robot speed**. As such, non-expected behavior (for example due to incorrect programming or wrong calibration) can be identified early enough to prevent the robot from colliding with surrounding objects or people.

Before running the program, it should be verified that **robot camera calibration** has been done correctly and that the **tool frame** has been defined correctly.

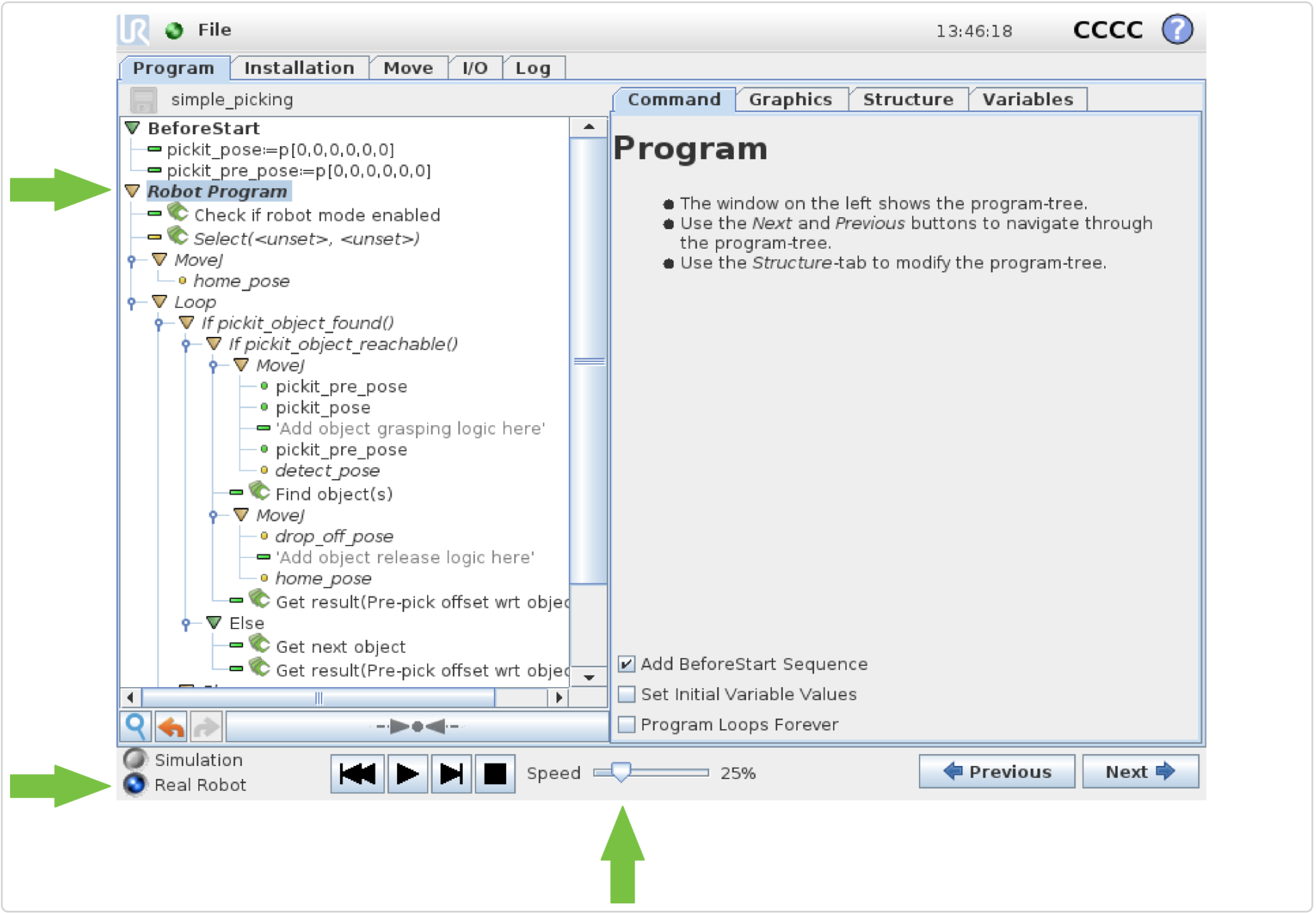
Please refer to the How to execute robot camera calibration (//support.pickit3d.com/article/35-how-to-execute-robot-camera-calibration) article for more details on how to perform robot camera calibration.

To allow Pick-it to respond to robot requests, Pick-it needs to be in **robot mode**. To enable robot mode, click on the following button on the Pick-it web interface:



In order to run the program in the robot controller, at the bottom of the graphical interface, make sure that **Real robot** is selected and that the robot **speed** is set to a safe value. Click on the rewind button to make sure that the program starts from the beginning - the program line **Robot Program** shall be highlighted, meaning that's the point at which the program will start. Finally, click on the play button to run the program.

The program execution can be stopped or paused by clicking on the stop and pause buttons respectively.



```
BeforeStart
  pickit_pose:=p[0,0,0,0,0,0]
  pickit_pre_pose:=p[0,0,0,0,0,0]
Robot Program
  Check if robot mode enabled
  Select(<unset>, <unset>)
  MoveJ
    home_pose
  Loop
    If pickit_object_found()
      If pickit_object_reachable()
        MoveJ
          pickit_pre_pose
          pickit_pose
          'Add object grasping logic here'
          pickit_pre_pose
          detect_pose
        Find object(s)
        MoveJ
          drop_off_pose
          'Add object release logic here'
          home_pose
        Get result(Pre-pick offset wrt objec
      Else
        Get next object
        Get result(Pre-pick offset wrt objec
```

Program

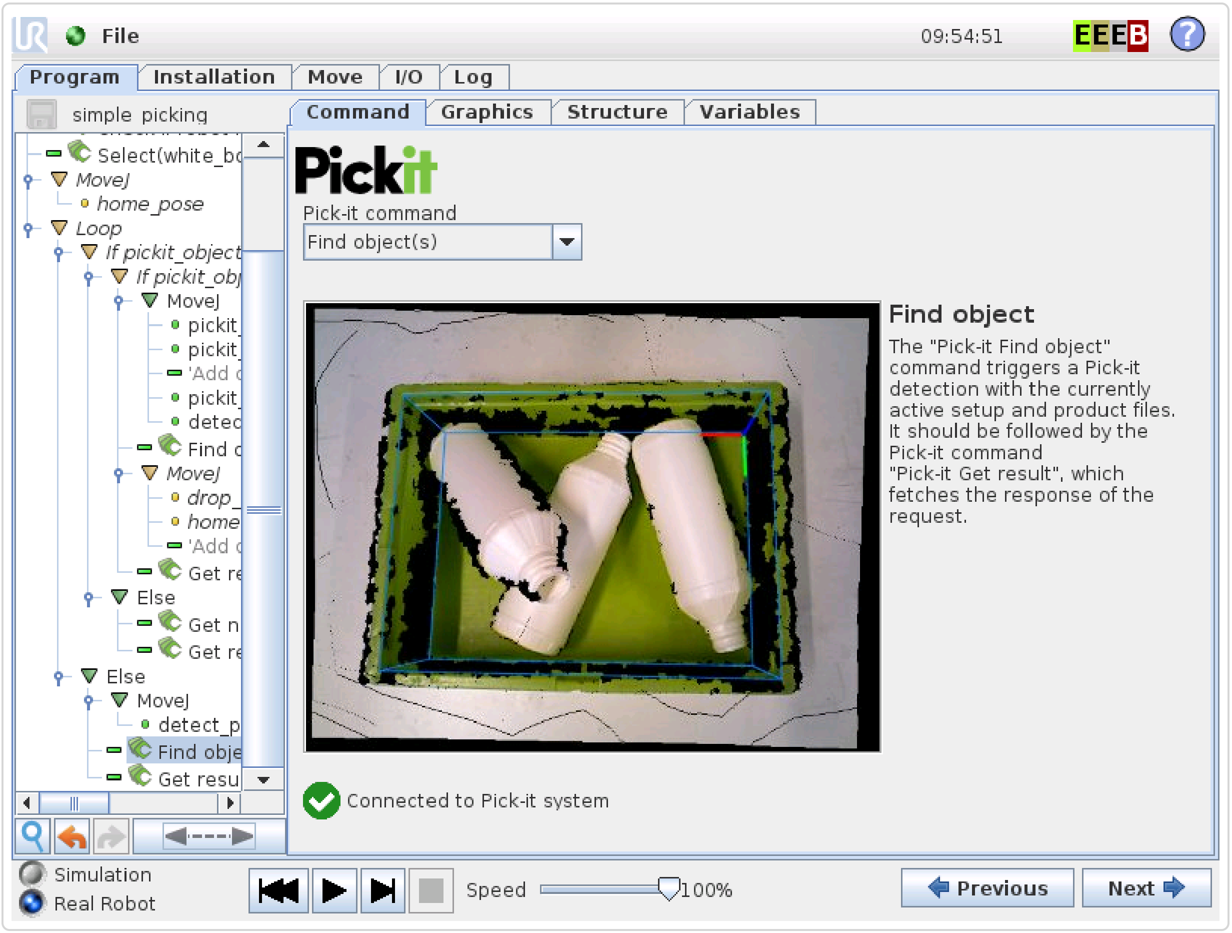
- The window on the left shows the program-tree.
- Use the *Next* and *Previous* buttons to navigate through the program-tree.
- Use the *Structure*-tab to modify the program-tree.

[✓] Add BeforeStart Sequence
[ ] Set Initial Variable Values
[ ] Program Loops Forever

Monitoring Pick-it from Polyscope

The Pick-it URCap plugin exposes the view of the Pick-it camera highlighted with the region of interest and detected object markers (the same as the 2D view in the Pick-it web interface). This is very convenient, as it allows to perform basic Pick-it monitoring without having to connect a separate computer to the Pick-it processor for opening the web interface.

To access the camera view, select any Pick-it URCap command on the robot program, and navigate to the `Command` tab on the right hand panel.



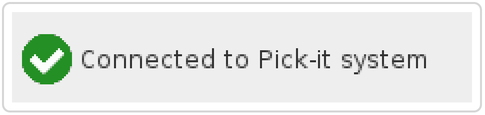
Pick-it command: Find object(s)

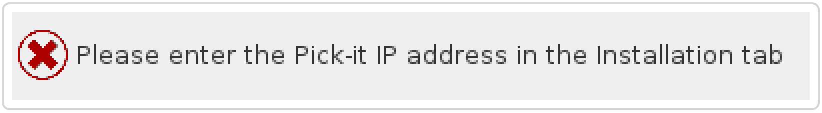
Find object

The "Pick-it Find object" command triggers a Pick-it detection with the currently active setup and product files. It should be followed by the Pick-it command "Pick-it Get result", which fetches the response of the request.

Connected to Pick-it system

Note that below the camera image there is an indicator of the connectivity with the Pick-it system that is constantly updated. This is the appearance of the indicator when Pick-it is connected and disconnected, respectively:


Connected to Pick-it system

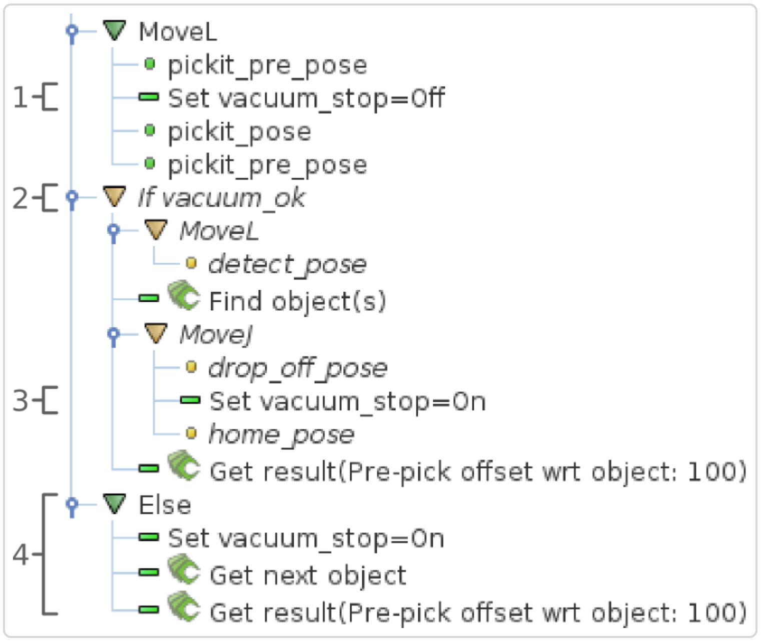

Please enter the Pick-it IP address in the Installation tab

Advanced topics

Gripping devices

The pick and place sequence presented in the main article omits details specific to the used gripping device. It also assumes a *perfect gripper*, i.e. that object picking always succeeds. This section presents a slightly more elaborate picking sequence for vacuum grippers that also handles failed picks. It introduces two signals that interact with the gripper hardware:

- `vacuum_stop` a digital output for controlling a normal-open valve that *enables suction* when the signal is *off / low*.
- `vacuum_ok` a digital input connected to a *vacuum sensor* used to detect pick success. When vacuum is expected but not present, the pick failed.



```
     MoveL
       pickit_pre_pose
1-[    Set vacuum_stop=Off
       pickit_pose
       pickit_pre_pose
2-[  If vacuum_ok
       MoveL
         detect_pose
       Find object(s)
       MoveJ
         drop_off_pose
3-[      Set vacuum_stop=On
         home_pose
       Get result(Pre-pick offset wrt object: 100)
     Else
       Set vacuum_stop=On
4-     Get next object
       Get result(Pre-pick offset wrt object: 100)
```

Noteworthy points of this pick and place sequence:

1. Vacuum is enabled at `pickit_pre_pose`, as suction cups typically have flexible bellows which allow initiating the grasp before reaching `pickit_pose`.
2. Next object detection and object drop-off are only performed if the pick was successful.
3. Vacuum is disabled to release object at `drop_off_pose`.
4. If object picking failed, disable vacuum and get the next available object, if any. If the next object is found and reachable, the next detection loop iteration will directly execute the picking sequence without incurring the overhead of moving back to `detect_pose` and waiting for a new detection to complete.

This pick and place sequence applies for the most part to other gripping devices where the pick action can be toggled with a digital output, and the pick success can be queried from a digital input. For instance, for a two-finger gripper the pick action would correspond to *closing the fingers*, and pick success could be *not reaching the end-of-stroke*. The only difference in the picking sequence would be that fingers close *after* reaching `pickit_pose`, as opposed to vacuum, which is enabled during the approach from `pickit_pre_pose`.

Robot mounted camera

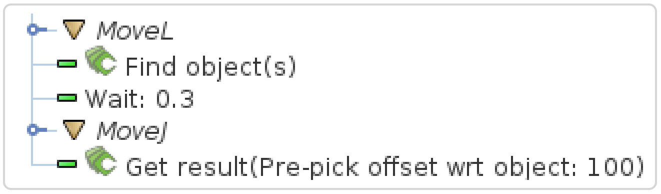
The example program assumes a fixed camera mount, but can be extended to robot mounted camera setups with minimal changes.

Detection pose

In a fixed camera setup `detect_pose` is such that the robot is not occluding the camera field of view. In a robot-mounted setup `detect_pose` must instead ensure that the camera is correctly pointing to the detection region.

Ensure image capture on a stationary robot

When the camera is mounted on the robot, it must be ensured that image capture is done on a stationary robot. If a robot motion starts just after calling `Find object(s)`, it is recommended to briefly wait to allow image capture to complete before starting to move. Image capture takes place only at the beginning of a Pick-it detection. In the example program, this only occurs once, in the object detected and reachable case.

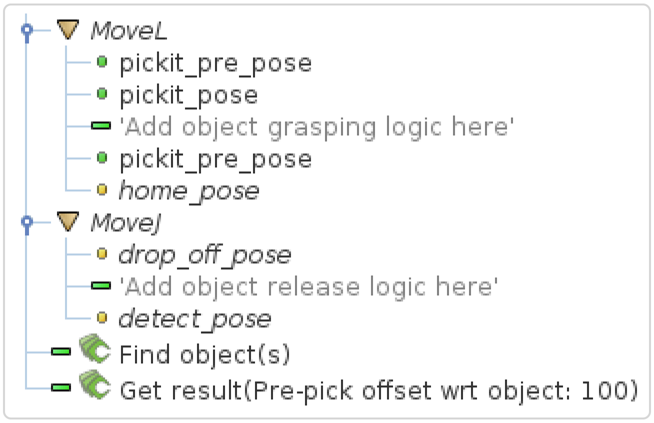


```
MoveL
  Find object(s)
  Wait: 0.3
MoveJ
  Get result(Pre-pick offset wrt object: 100)
```

When `Find object(s)` is immediately followed by `Get result`, an explicit `Wait` statement is not required.

Cycle time optimization

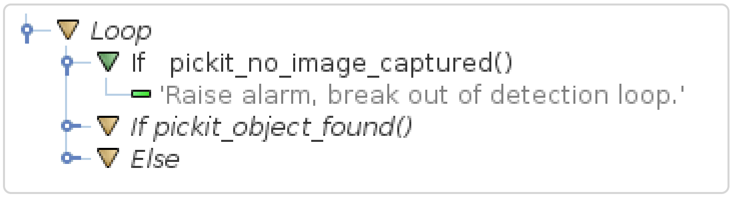
The optimization implemented in the pick sequence triggers the next object detection while the current object is being grasped. Depending on the camera fixture, the gripper and the object geometry; it might be the case that the grasped object occludes the camera field of view, making it impossible to trigger a detection with a grasped object. When this is the case and hardware modifications are not an option, the optimization cannot be done. The non-optimized pick and place sequence then becomes:



```
MoveL
  pickit_pre_pose
  pickit_pose
  'Add object grasping logic here'
  pickit_pre_pose
  home_pose
MoveJ
  drop_off_pose
  'Add object release logic here'
  detect_pose
Find object(s)
Get result(Pre-pick offset wrt object: 100)
```

Monitoring camera disconnections

As part of application monitoring, you can add a `pickit_no_image_captured()` check that verifies if object detection was unsuccessful due to a failure to capture a camera image. When this is the case, it typically indicates a a hardware disconnection issue, such as a loose connector or broken cable. This function can be used to send an alarm to a higher level monitoring system that an operator can quickly respond to. This is how the detection loop would look with the added check:



```
Loop
  If  pickit_no_image_captured()
    'Raise alarm, break out of detection loop.'
  If pickit_object_found()
  Else
```

Still need help? Contact Us (/contact)

Last updated on March 6, 2018