Fakultät
Informatik und Mathematik

User manual for RobotController.dll

Sommersemester 2020

Nikola Bergmaier

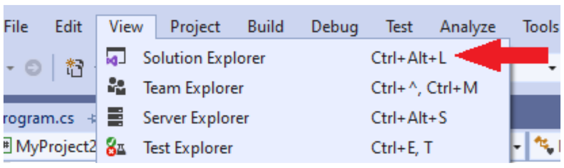
Contents

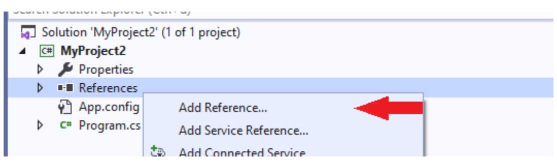
1 How to import the DLL in a project

The integration of a DLL can be different for each IDE. This manual will only show how to integrate the RobotController.dll in Visual Studio Community.

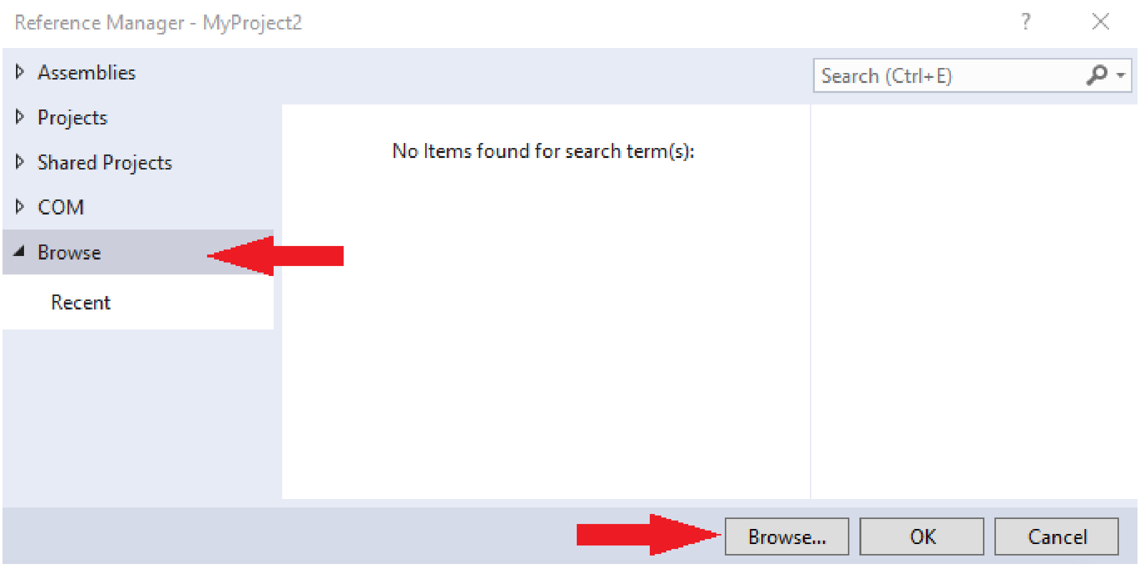
First make sure that you have the complete RobotController.dll folder with the RobotController.dll AND the RobotController.xml file!

After creating a Console App project in .NET Framework, open the solution explorer, right click on “References” and click “Add Reference...”.
If the Console App runs on .NET Core, open the solution explorer, right click on “Dependencies” and click “Add Project Reference...”.

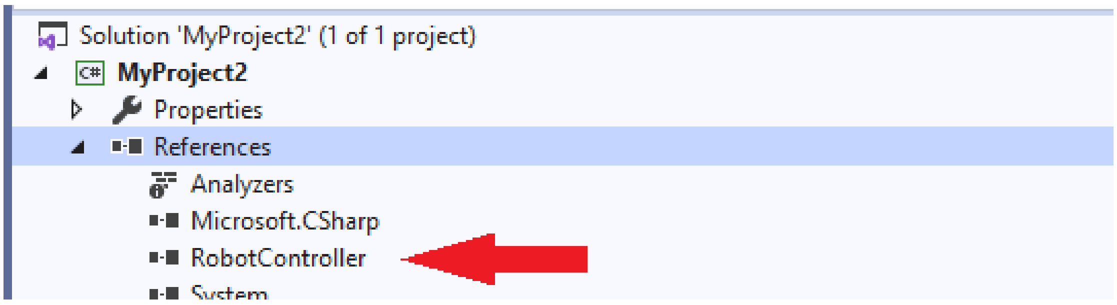




Now choose the “Browse” tab on the left and click on the “Browse...” button at the bottom right.



Search the RobotController.dll file and click “OK”. You should now see RobotController listed under “References” in the Solution Explorer.
With .NET Core you should see RobotController listed under “Dependencies” → “Assemblies”.



2 How to use the DLL

First import the namespace RobotController, then create an object of the class RobotControl. This object gives you all the functionality you need to control the robot.
Start by connecting to the robot, then use your commands and close the connection at the end.

```
using RobotController;

namespace MyProject
{
    class Program
    {
        static void Main(string[] args)
        {
            RobotControl control = new RobotControl();
            control.StartConnection();
            //enter commands here
            control.CloseConnection();
        }
    }
}
```

When working with the robot use the code above. If you only work with the Robot-Simulation project and not with the robot, you have to call the constructor of RobotControl with the IP-address “127.0.0.1” and the port “59152”.

3 Command overview

3.1 Home motion

The robot has a predefined idle position. You can move to this position with the following command.

```
RobotPosition pos = control.MoveToHome();
```

Each command returns the current position of the robot in the current base. The RobotPosition class contains the cartesian coordinates in mm (RobotCartPosition), the axis values in degree (RobotAxisPosition) and the gripper status (bool).

3.2 Motion with cartesian target position

The robot can move to a given cartesian position. This position is an object of the class RobotCartPosition.

```
RobotPosition pos = control.MoveTCPToPosition(new RobotCartPosition
    (10,20,30,0,0,90), RobotCartMoveType.LIN, false);
```

When creating a RobotCartPosition object, you can set the x, y, z coordinates and optionally the orientation of the target position. If the motion is a PTP-motion you can add the status for the position as well.
The other parameters specify the motion type (PTP or LIN) and if the coordinates are relative to the current position or absolute.

3.3 Motion with axis-specific target position

The robot can move to a given axis position with a PTP motion. This position is an object of the class RobotAxisPosition.

```
RobotPosition pos = control.MoveAxesToPosition(new RobotAxisPosition
    (10,20,30,40,50,60), false);
```

When creating a RobotAxisPosition object, you have to set all the axis values.
The second parameter decides if the values are relative to the current position or absolute.

3.4 Gripper control

The robot can open or close the gripper.

```
RobotPosition pos = control.CloseGripper();
RobotPosition pos = control.OpenGripper();
```

3.5 Workspaces

The robot has certain workspaces in which he can move. One of them is a cartesian space, one of them contains ranges for each axis.

You can get both workspaces as properties.

```
AxisSpace axisSpace = control.AxisWorkspace;
CartSpace cartesianSpace = control.CartWorkspace;
```

A AxisSpace has a minimum and maximum value for each axis.
A CartSpace has a minimum and maximum value for each coordinate and a origin frame, to which the limits refer.

If you call a motion command, which violates the limits of a workspace, the program will throw an InvalidMovementException.

Forbidden workspaces The user can define additional workspaces, that are forbidden for the robot.

```
Dictionary<string, AxisSpace> axisSpaces = control.ForbiddenAxisSpaces;
Dictionary<string, CartSpace> cartSpaces = control.ForbiddenCartSpaces;

// work directly with the Dictionary
cartSpaces.Add("myName", new CartSpace((10, 10, 10), (50, 50, 50)));
cartSpaces.Remove("myName");

// use the methods of RobotControl
control.AddForbiddenSpace("myName", new CartSpace((10, 10, 10), (50,
    50, 50)));
control.RemoveForbiddenSpace("myName");
```

When creating a cartesian workspace you can optionally set an origin frame to which the limits refer. If you don't set a frame, the space has the world coordinate system as origin frame.
When adding or removing such workspaces, you can either work with the dictionaries the RobotControl offers as properties, or use predefined methods provided by the RobotControl.

3.6 Base and Tool control

The robot has a base and a tool coordinate system, which can be set by the user.

```
RobotFrame @base = control.Base;
RobotFrame tool = control.Tool;

IReadOnlyDictionary<int, RobotFrame> base_data = control.Base_Data;
IReadOnlyDictionary<int, RobotFrame> tool_data = control.Tool_Data;

RobotPosition pos = control.SetBase(new RobotFrame(10, 10, 10, 0, 0,
    90));
RobotPosition pos = control.SetBase(4);
```

When setting the base/tool you can use a custom frame or use an index for the Base_Data/Tool_Data array. These arrays are provided by the RobotControl as properties.
After the user sets the base/tool, the methods returns the current position after the transformation with the new base/tool.

3.7 System variables control

The robot has system variables, like the base and tool coordinate system, analog outputs and many more. Since not all of these system variables should be changeable by the user, only the allowed ones can be set with the DLL.

```
    control.SetSystemVariable("BASE", new RobotFrame(10, 10, 10, 0, 0,
        90));
    control.SetSystemVariable("ANOUT", 0.5f, 1);
```

When setting the system variable, you declare the name of the variable and the new value (should match the type of the system variable). There are system variables, like analog outputs (AnOut), which are given as an array. For these cases you can optionally set an index. In the example above, we would set the analog output AnOut[1] to 0.5.

3.8 Rotation table control

The robot at the OTH Regensburg has a rotating table connected to the robot, which can be controlled with the DLL.

```
    control.RotateTable(180);
    control.RotateTable(-90, true);
```

The table starts at the angle 0. To rotate it, you can set the angle in degree. Optionally you can define the rotation relative to the current position.