KINOVA

Matlab Wrapper for Kinova API

Adding a function of the Kinova API to the Matlab Wrapper.

First, make sure that it is declared in the CommandLayer.h header file. If it is not declared in this file, it means that this is not a Kinova API function.

If the function does not take any parameters

This section is made for the function that does not take any parameters and you just want to call it without adding anything. If it is your case, follow the next steps.

- At the top of the JacoSDKWrapper.cpp file, declare a pointer to the function that you want to use .
  ```
  int (*My<FunctionName>)();
  ```

- A little below, in the openKinovaLibrary function, assign the pointer that you just declared before.
  ```
  My<FunctionName> = (int(*)()) GetProcAddress(commandLayer_handle,
  "<FunctionName>");
  ```

- At the end of the JacoSDKWrapper.cpp file, create a function that call your pointer to the <FunctionName>.

  ```
  bool <functionName>()
  {
          if (MyInitAPI != NULL && commandLayer_handle != NULL &&
          My<FunctionName> !=NULL)
          {
                  My<FunctionName>();
                  return true;
          }

          else
          {
                  mexPrintf("Library or API not open\n");
                  return false;
          }
  }
  ```

NOTE : Make sure that the function you create returns a bool.

- In the JacoSDKWrapper.h header file, add the function indice in the FunctionIndice class and the function header as an extern C.
  ```
  k<FunctionName> = id,
  extern "C" bool <functionName>();
  ```

- In the JacoMexInterface.cpp file, at the end, add a new case to call the function you just created.
  ```
  case FunctionIndices::k<FunctionName>:
          *mxGetLogicals(plhs[0]) = set<FunctionName>();
          break;
  ```

- In the MexFunctionsIDs.m, add the function id that you have created in the JacoSDKWrapper.h file.
  ```
  FUNCT_TAG(id)
  ```

  NOTE : the FUNCT_TAG can be whatever you want but the id must be the same as the one you set in the JacoSDKWrapper.h file.

- In the JacoComm.m file, add the definition of the function as a public method. You have the use the following template to call the cpp function that you have created.
  ```
  function <FunctionName>(obj)
          stat = false;
          if coder.target('MATLAB')
            [stat,~,~,~,~,~,~,~,~] =
  JacoMexInterface(double(MexFunctionIDs.FUNCT_TAB),..
  obj.JntPosCmd,obj.JntVelCmd,obj.JntTorqueCmd,obj.FingerPosCmd,obj.CartPosCmd,
  obj.OffsetCmd);
          else
            stat = coder.ceval('FunctionName');
          end
          if ~stat
            error('Failed to start FunctionName');
          end
      end
  ```

- Now you have to recompile the Mexfiles. In the compileMexInterface.m file, change the path of each file and directory to yours. Leave the last line as it is.
  When it's done, you have to run the compileMexInterface.m file and close Matlab.

- Now in the testJacoComm.m file, you can call the function you have created in the JacoComm.m file. If you follow this guide step by step, it should be something like that: jc.<FunctionName>

- You can now call the function by running the section of testJacoComm.m where you call the function.

If the function takes input parameters

Taking input parameters means that the function must use some information. Basically, it is a “set” function. If it is your case, follow the next steps.

- At the top of the JacoSDKWrapper.cpp file, declare a pointer to the function that you want to use .
  int (*My<FunctionName>)(<argument>);

  NOTE : Make sure you declare the argument the same way as it define in the CommandLayer.h header file.

- A little below, in the openKinovaLibrary function, assign the pointer that you just declared before.
  My<FunctionName> = (int(*)(<argument>)) GetProcAddress(commandLayer_handle, "<FunctionName>");

- At the end of the JacoSDKWrapper.cpp file, create a function that call your pointer to the <FunctionName>.

```
bool <functionName>(arg)
{
        if (MyInitAPI != NULL && commandLayer_handle != NULL &&
        My<FunctionName> !=NULL)
        {
                My<FunctionName>(arg);
                return true;
        }

        else
        {
                mexPrintf("Library or API not open\n");
                return false;
         }
}
```

  NOTE :The arg, is the argument that you want to pass from Matlab to CPP source files. In this example, it could be a position that we want to send to the CPP files. Also note that the function have to return a bool.

- In the JacoSDKWrapper.h headerfile, add the function indice in the FunctionIndice class and the function header as an extern C.
  ```
  k<FunctionName> = id,
  extern "C" bool <functionName>(<argument>);
  ```

- In the JacoMexInterface.cpp file, in the mexFunction, declare an input pointer that represents what you want to send.
  ```
  double * in<Argument>;
  ```

- A few lines below, in the following line, add one to the nrhs condition and adapt the error message.
  ```
  if(nrhs!=7) { // (fcnIndex, jntPos, jntVel, jntTorque, fingerPos, cartPos)
      mexErrMsgIdAndTxt("MyToolbox:nrhs","7 inputs required.");
  }
  ```

- Below again, Check that the input is the right dimensions. You have to edit the yellow part to fit the argument that you are going to pass.
  ```
  /* Seventh Input: Offset: Check that number of rows and cols */
    if(mxGetM(prhs[6]) != 4 || mxGetN(prhs[5]) != 1) {
      mexErrMsgIdAndTxt("MyToolbox:notColVector",
                "Offset command Input must be a Col vector of 4 elements.");
    }
  ```
  NOTE : The prhs array starts at 0, so this is not the same number as the step before.

- Below again, assign the input pointer to the mex input.
  ```
  in<Argument> = mxGetPr(prhs[6]);
  ```

- At the end of JacoMexInterface.cpp file, add a new case to call the function you just created.
  ```
  case FunctionIndices::k<FunctionName>:
        *mxGetLogicals(plhs[0]) = <FunctionName>();
        break;
  ```

- In the MexFunctionsIDs.m, add the function id that you have created in the JacoSDKWrapper.h file.
  ```
  FUNCT_TAG(id)
  ```

  NOTE : the FUNCT_TAG can be whatever you want, but the id must be the same as the one you set in the JacoSDKWrapper.h file.

- In the JacoComm.m file, add a private property of the argument that you want to send to your function and initialise it to zero.
  ```
  <Argument>Cmd = zeros(4,1);
  ```

- In the JacoComm.m file, add the definition of the function as a private method. you have to use the following template.
  ```
  function <FunctionName>Internal(obj,Cmd)
        stat = false;
        obj.<Argument>Cmd = Cmd;
        if coder.target('MATLAB')
          [stat,~,~,~,~,~,~,~,~] =
  JacoMexInterface(double(MexFunctionIDs.FUNCT_TAG),...
  obj.JntPosCmd,obj.JntVelCmd,obj.JntTorqueCmd,obj.FingerPosCmd,obj.CartPosCmd,
  obj.OffsetCmd,obj.<Argument>Cmd);
        else
          stat = coder.ceval('<FunctionName>');
        end
        if ~stat
          error('Failed to set <FunctionName>');
        end
      end
  ```

- Once you have done that, you have to add obj.<Argument>Cmd every time JacoMexInterface is called in the JacoComm.m file.
  ```
  [stat,~,~,~,~,~,~,~,~] =  JacoMexInterface(double(MexFunctionIDs.SET_EE_OFFSET),...
  obj.JntPosCmd,obj.JntVelCmd,obj.JntTorqueCmd,obj.FingerPosCmd,obj.CartPosCmd,
  obj.OffsetCmd,obj.<Argument>Cmd);
  ```

- In the JacoComm.m file, add a public method to check if the arguments are valid and to call the private function. You have the use the following template.
  ```
  function <FunctionName>(obj,cmd)

        % Validate input
        validateattributes(cmd, {'numeric'},...
          {'real', 'nonnan','nonempty','finite', 'column',...
          'nrows',4},'<FunctionName>');

        % Call private function
        obj.<FunctionName>Internal(cmd);
      end
  ```
  NOTE : Change the yellow part to fit your argument dimensions.

- Now you have to recompile the Mexfiles. In the compileMexInterface.m file, change the path of each file and directory to yours. Leave the last line as it is.
  When it's done, you have to run the compileMexInterface.m file and close Matlab.

- Now in the testJacoComm.m file, you can call the function you have created in the JacoComm.m file. If you follow this guide step by step, it should be something like that:

  ```
  <Argument>Cmd = [value]
  jc.<FunctionName>(<Argument>Cmd);
  ```

- You can now call the function by running the section of testJacoComm.m where you call the function.

If the function takes output parameters

Taking outputs parameters means that the function must return you some information. Basically, it is a “get” function. If it is your case, follow the next steps.

- At the top of the JacoSDKWrapper.cpp file, declare a pointer to the function that you want to use .
  int (*My<FunctionName>)(<argument>);

  NOTE : Make sure you declare the argument the same way as it define in the CommandLayer.h header file.

- A little below, in the openKinovaLibrary function, assign the pointer that you just declared before.
  My<FunctionName> = (int(*)(<argument>)) GetProcAddress(commandLayer_handle, "<FunctionName>");

- At the end of the JacoSDKWrapper.cpp file, create a function that call your pointer to the <FunctionName>.

```
bool <functionName>(arg)
{
        if (MyInitAPI != NULL && commandLayer_handle != NULL &&
        My<FunctionName> !=NULL)
        {
                My<FunctionName>(arg);
                return true;
        }

        else
        {
                mexPrintf("Library or API not open\n");
                return false;
        }
}
```

  NOTE :The arg, is the argument that you want to pass from Matlab to CPP source files. In this example, it could be a position that we want to get from the .cpp files. Also note that the function have to return a bool.

- In the JacoSDKWrapper.h headerfile, add the function indice in the FunctionIndice class and the function header as an extern C.
  k<FunctionName> = id,
  extern "C" bool <functionName>(<argument>);

- In JacoMexInterface.cpp file, to get information from your function, you have to declare an output pointer that represents what you want to get.
  type *out<Argument>;

- A few lines below, in the following line, add one to the nrhs condition and adapt the error message.

```
if(nlhs!=9) { //[status, pos, vel, torque, temp, ee_pose,ee_wrench, DOF]
    mexErrMsgIdAndTxt("MyToolbox:nlhs","9 outputs required.");
  }
```

- Below again, create the output value for your output pointer.

```
plhs[8] = mxCreateDoubleMatrix(4,1,mxREAL);
  out<Argument> = mxGetPr(plhs[8]);
```

NOTE : The prhs array starts at 0, so this is not the same number as the step before.

- At the end of JacoMexInterface.cpp file, add a new case to call the function you just created.

```
case FunctionIndices::k<FunctionName>:
      *mxGetLogicals(plhs[0]) = set<FunctionName>();
      break;
```

- In the MexFunctionsIDs.m, add the function id that you have created in the JacoSDKWrapper.h file.
  FUNCT_TAG(id)

  NOTE : the FUNCT_TAG can be whatever you want, but the id must be the same as the one you set in the JacoSDKWrapper.h file.

- In the JacoComm.m file, add a protected property of the argument that you want to send to your function and let it uninitialized.
  <Argument>;

- In the JacoComm.m file, add the definition of the function as a private method. You have to use the following template.
  function arg = get<Argument>(obj)

```
        stat = false;
        argument = zeros(4,1);
        if coder.target('MATLAB')
          [stat,~,~,~,~,~,~,~,argument] =
JacoMexInterface(double(MexFunctionIDs.FUNCT_TAG),...
obj.JntPosCmd,obj.JntVelCmd,obj.JntTorqueCmd,obj.FingerPosCmd,obj.CartPosCmd,
obj.OffsetCmd);
        else
          stat = coder.ceval('get<FunctionName>',coder.wref(argument));
        end
        if ~stat
          error('Failed to get argument');
        end
        arg = argument;
      end
```

- Defines a public function to call the private function you just created.

```
 function arg = get.<Argument>(obj)
        if obj.IsConnected
          arg = obj.get<Argument>;
        else
          arg = [];
        end
      end
```

- Once you've done that, you have to add <,~> every time JacoMexInterface is called in the JacoComm.m file.

```
[stat,~,~,~,~,~,~,~,~,~] =
JacoMexInterface(double(MexFunctionIDs.SET_EE_OFFSET),...
obj.JntPosCmd,obj.JntVelCmd,obj.JntTorqueCmd,obj.FingerPosCmd,obj.CartPosCmd,
obj.OffsetCmd);
```

- Now you have to recompile the Mexfiles. In the compileMexInterface.m file, change the path of each files and directory to yours. Leave the last line as it is.
  When it's done, you have to run the compileMexInterface.m file and close Matlab.

- Now in the testJacoComm.m file, you can call the function you have created in the JacoComm.m file. If you follow this guide step by step, it should be something like that:
  jc.<Argument>

- You can now call the function by running the section of testJacoComm.m where you call the function.

Simulink implementation

How to create a Simulink block with the JacoComm class

- In Simulink, **View > Library Browser**.
- In Library Browser, search for MATLAB System and drag and drop to your project.
- Double click on the MATLAB system block and browse for JacoComm.m file. [YOUR_PATH]\matlab_Kinovaapi_wrapper-master\JACO2Communicator\Source
- Select OK

How to add an output port to the JacoComm block

- In JacoComm.m file, in the protected methods section, in stepImpl function, assign to a variable the value you want. By example, Fee = Obj.getEndEffectorWrench();
- In isOutputFixedSizeImpl function, add varargout{n+1} = true;
- In getOutputSizeImpl function, add one output named o(n+1) and assign it to the dimension you want.
- In isOutComplexImpl function , add varargout{n+1} = false;
- In getOutputDataTypeImpl function, add varargout{n+1} = double;

How to add an input port to the JacoComm block

- In JacoComm.m file, in the protected methods section, in getNumInputsImpl, change num = n to num = n+1
- In getInputNamesImpl, add varargout{n+1} = '<NameOfTheInputPort>'
- You can validate it in the validateInputsImpl function (see other validation in this function).

- You can process you input port in the
- stepImpl function. Use varargin{n+1}