# UNIVERSIDAD DE TARAPACÁ

## FACULTAD DE INGENIERÍA

Departamento de Ingeniería en Computación e Informática

# Formulación del Proyecto
# Arturi-toe

**Autor(es): Javier Mamani**

**Mauricio Mamani**

**Julio Rivera**

**Rodrigo Carvajal**

**Sebastian Lukich**

**Asignatura: Proyecto l**

**Profesor(es): Ricardo Valdivia**

ARICA, 17 OCTUBRE DEL 2019

## Historial de Cambios

| Fecha | Versión | Descripción | Autor(es) |
|---|---|---|---|
| 13/08/2019 | 1.0 | Enfoque de ideas para crear un robot capaz de cumplir con cada función requerida. | Julio Rivera<br>Javier Mamani |
| 20/08/2019 | 1.1 | Versión del robot con garra que abre y cierra y motor mediano que la hace girar. Con problemas de equilibrio. | Rodrigo Carvajal |
| 27/08/2019 | 1.2 | Versión del robot con motor grande que hace girar la garra, que abre y cierra. Problemas de equilibrio casi completamente eliminados. | Rodrigo Carvajal |
| 03/09/2019 | 1.3 | Nuevo armado del robot con un brazo y una base más estables. Estructura de la garra arreglada. Además de un entorno a programar más adecuado y amigable. | Rodrigo Carvajal<br>Julio Rivera |
| 12/09/2019 | 1.4 | El robot tiene interfaz gráfica para sus controles. El manejo de cables y uso de la pantalla del brick se han facilitado | Mauricio Mamani<br>Julio Rivera |
| 08/10/2019 | 1.5 | Se cambió en el código la dirección de los giros. Se cambió un cable del robot por otro más corto en otra posición | Rodrigo Carvajal |

# Tabla de Contenidos

7. Resultados
   7.1 Estado actual del proyecto
   7.2 Problemas encontrados y soluciones propuestas
   7.3 Conclusiones

8. Referencias (utilizando el estándar IEEE)
   Anexos
   Anexo A: Hardware (diagrama de construcción del robot, componentes principales)
   Anexo B: Software (código de los programas implementados)
   Anexo C: Comunicaciones (configuración de comunicación pc-robot considerando RPyC)

# ANÁLISIS DE DISEÑO

## Especificación de requerimientos

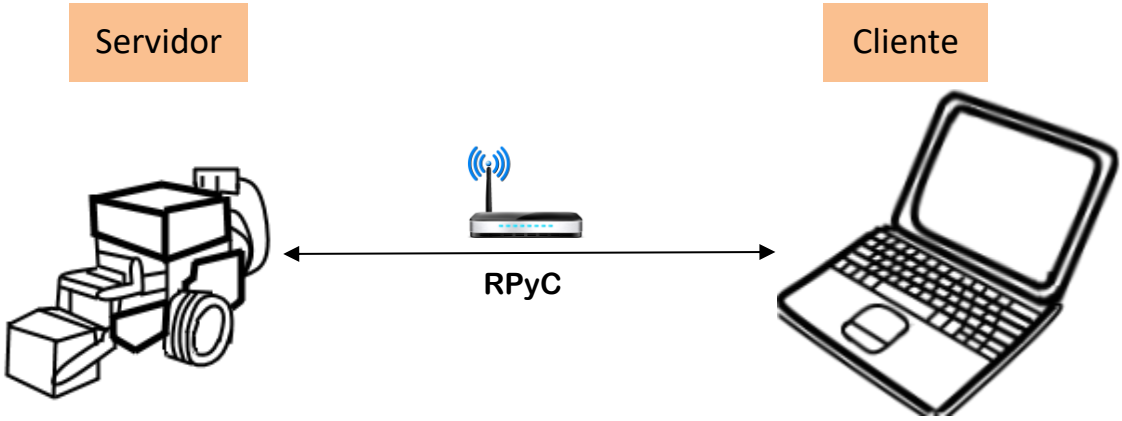
**Funcionales**

a. Se requiere que el robot se acerque a una lata, la alce y pueda llevársela como carga.
b. Se requiere que el robot voltee la lata que carga consigo y la suelte en posición invertida en el suelo.

**No funcionales**

a. Se requiere que el código del robot sea diseñado en lenguaje Python.
b. Se requiere que el robot sea armado sólo usando piezas Lego.
c. Se requiere que la lata del juego sea el modelo de lata de bebida más pequeño que se encuentre a la venta.
d. Se requiere que el robot no tenga problemas de equilibrio.

Arquitectura propuesta



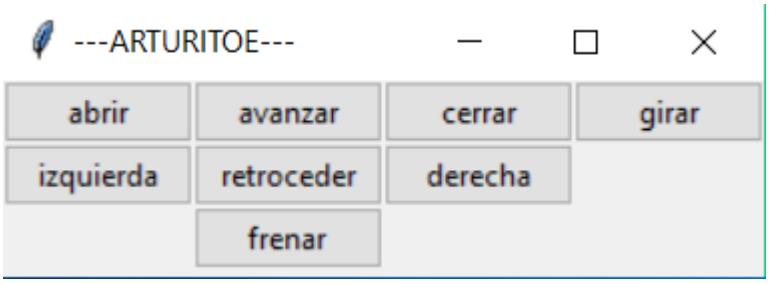

## Software Movimiento:

- Método de abrir y cerrar garra
- Método de hacer girar la garra
- Método de decir mensaje de voz
- Método de avanzar hacia donde mira el robot
- Método de retroceder
- Método de girar garra a la derecha
- Método de girar garra a la izquierda
- Método de frenar robot

## Software Interfaz:

- Ventana título Arturi-toe
- Botón texto “avanzar”
- Botón texto “retroceder”
- Botón texto “izquierda”
- Botón texto “derecha”
- Botón texto “frenar”
- Botón texto “abrir”
- Botón texto “cerrar”

## Diseño de la interfaz de usuario




El usuario dispondrá de una interfaz como la que se muestra en la imagen, para poder controlar y comunicarse remotamente con el robot. El funcionamiento de los botones es:

1. Abrir: El robot abre su garra mientras se mantenga presionado el botón.
2. Avanzar: El robot avanza mientras se mantenga presionado el botón.
3. Cerrar: El robot cierra su garra mientras se mantenga presionado el botón.
4. Girar: EL robot gira su garra 180° grados.
5. Izquierda: El robot gira en su propio eje hacia la izquierda mientras se mantenga presionado el botón.
6. Retroceder: El robot retrocede mientras se mantenga presionado el botón.
7. Derecha: El robot gira en su propio eje hacia la derecha mientras se mantenga presionado el botón.
8. Frenar: Detiene todos los procesos que están en marcha.

# RESULTADOS

## 1. Estado actual del proyecto

Los movimientos del robot se controlan única y completamente por el ordenador, a través de una interfaz gráfica hecha con software Python. La garra rota, se abre y cierra por comandos de la interfaz y es capaz de tomar la lata de aluminio y girarla. El robot logra avanzar mientras se presiona el botón de la interfaz destinado a esa función. Asimismo, al soltarlo, se frena; no obstante, hay un botón de frenado redundante. Además, el robot logra girar a la derecha, izquierda, retroceder de manera similar a como funciona el control de avance hacia adelante.

El código en Python actual permite no sólo el disponer de una interfaz gráfica sino también unos controles más intuitivos para el usuario. Cada control tiene asociada una tecla diferente del teclado, con el añadido de que se pueden presionar los botones en pantalla para hacer las mismas funciones que hace ya el teclado. Entre estas funciones: avanzar, retroceder, girar a la izquierda, girar a la derecha, dejar la garra en modo abrir, dejar la garra en modo cerrar, detener la apertura/ cierre de la garra de golpe y, girar la garra.

La conexión que se usa para conectar al cliente -la computadora- con el servidor -el robot- es vía Wifi.

El robot ya se ha probado en un entorno parecido al de Flip-tac-toe con latas y otros robots en el campo.

Existe un manual de usuario que hasta ahora sólo explica los componentes físicos del robot y sus funciones.

Existe una wiki que incluye desde fotos del robot armado en su totalidad hasta una explicación de los componentes de su interfaz. Incluye introducción, objetivos generales, objetivos específicos, gestión de los gastos, riesgos, entre otras cosas que se desarrollan también en el informe.

El informe del proyecto incluye introducción, objetivos generales, objetivos específicos, actividades, gestión de los gastos, riesgos, carta Gantt; y ahora último se ha añadido análisis de diseño, resultados, implementación y más referencias en forma de anexos.

2. Problemas encontrados y soluciones propuestas

Si bien la garra desempeña su función de tomar la lata y girarla aún existe dificultad en la tarea de frenar al robot frente a la lata sin chocarla ni tumbarla. Este hecho está conectado al hecho de que al soltar el botón de avance de la interfaz el robot sigue avanzando por unos momentos (menos de 1s). Para hacer frente a esto, se propone resolver los retrasos en la conexión a internet usando no la red Wifi común del laboratorio donde se hará la competencia, sino una red propia del equipo compartida por uno de sus miembros, usando su plan de datos. Otra alternativa sería implementar un botón nuevo de avanzar lento, dejando la velocidad por defecto del robot en rápida, como ya estaba configurada.

3. Conclusiones

- El robot recibe órdenes de una computadora vía una red wifi. Esto bien lo limita a la velocidad o conexión que le brinda la red y podría mejorarse con una red especial sólo para el robot.
- El robot desempeña satisfactoriamente sus funciones de moverse a lo largo del suelo, agarrar la lata de bebida y girarla. No quedan otros requerimientos funcionales por añadir en los objetivos de este proyecto.
- El robot posee conexión a una interfaz en la computadora que permite de manera intuitiva dar órdenes a cada una de las funciones de movimiento, funciones de uso de la garra. El retraso que se observa al presionar un botón de la interfaz y medir el tiempo de reacción del motor del robot se puede aliviar mejorando la conexión de Wifi.
- El espacio que necesitaría el robot en promedio para entrar al tablero de Flip-tac-toe y tomar una lata es de tres baldosas del suelo que recubre el laboratorio donde se hicieron las pruebas, esto es, alrededor de dos veces el ancho del robot del equipo. El tablero que se adecúa por convenio es de 7x7 baldosas con una lata en cada esquina, en el centro de cada lado y en el centro del tablero.

# IMPLEMENTACIÓN

A través de la herramienta pydoc “python -m pydoc prueba2”

```
NAME
        prueba2

CLASSES
        rpyc.core.service.Service(builtins.object)
        Robot

        class Robot(rpyc.core.service.Service)
        |  clase robot.
        |  el robot tiene como atributos
        |  nombre, un booleano "giro" (utilizado en la funcion para girar la garra),
        |  un booleano "avanzando" (utilizado en las funciones para girar)
        |  y 4 objetos de motores, 3 motores grandes (rueda_izq, rueda_Der y girar garra),
        |  utilizados para las 2 ruedas y para hacer el giro de la garra, y cuenta tambien con
        |  un motor mediano, utilizado para abrir y cerrar la garra
        |
        |  Method resolution order:
        |          Robot
        |          rpyc.core.service.Service
        |          builtins.object
        |
        |  Methods defined here:
        |
        |  exposed_Abrir(self)
        |          Funcion para abrir la garra.
        |
        |          esta funcion activa el motor a mitad de potencia para
        |          abrir la garra, hasta que se llame al metodo "frenar"
        |
        |  exposed_Cerrar(self)
        |          Funcion para cerrar a garra.
        |
        |  exposed_Avanzar(self)
        |          funcion para que el robot avance.
        |
```

```
        |       esta funcion activa los 2 motores de las ruedas, estas se mantendran
        |       girando, hasta que se llame a la funcion "frenar"
        |
        | exposed_Frenar(self)
        |       funcion para frenar el robot.
        |
        |       esta funcion detiene los motores de las ruedas y el motor que abre y cierra la garra
        |       ademas deja la variable booleana "avanzando" en false, asi al momento de girar,
        |       este girara como si estubiera detenido.
        |
        | exposed_Girar_der(self)
        |       funcion para girar a la derecha.
        |
        |       esta funcion hace girar solo una rueda con el fin de que el robot gire
        |       usa la variable booleana "avanzando", para saber si el robot se encuentra en
movimiento,
        |       ya que, se cambia la velocidad de la rueda a una velocidad menor que en las
funciones
        |       "avanzar" y "retroceder", asi que se debe cambiar la velocidad de la rueda
contraria al
        |       lado donde se desea girar
        |
        | exposed_Girar_izq(self)
        |       funcion para girar a la izquierda.
        |
        |       esta funcion hace girar solo una rueda con el fin de que el robot gire
        |       usa la variable booleana "avanzando", para saber si el robot se encuentra en
movimiento,
        |       ya que, se cambia la velocidad de la rueda a una velocidad menor que en las
funciones
        |       "avanzar" y "retroceder", asi que se debe cambiar la velocidad de la rueda
contraria al
        |       lado donde se desea girar
        |
        | exposed_Giro_Garra(self)
        |       Funcion para hacer el giro de la garra.
        |
```

```
    |       esta funcion ocupa la variable "giro", para saber si ya se giro la garra anteriormente,
    |       con el fin de que no se enrede el cable del motor que abre y cierra la garra.
    |       el motor se activa por un tiempo y velocidad determinada, para que haga el giro en 180°
    |
    | exposed_Retroceder(self)
    |       funcion para que el robot retroceda.
    |
    |       esta funcion activa los 2 motores de las ruedas con una velocidad
    |       negativa, para asi retroceder, estas se mantendran girando, hasta
    |       que se llame a la funcion "frenar"
    |
    | ----------------------------------------------------------------------
    | Data descriptors defined here:
    |
    | __dict__
    |       dictionary for instance variables (if defined)
    |
    | __weakref__
    |       list of weak references to the object (if defined)
    |
    | ----------------------------------------------------------------------
    | Data and other attributes defined here:
    |
    | avanzando = False
    |
    | giro = True
    |
    | nombre = 'arturitoe'
    |
    | ----------------------------------------------------------------------
    | Methods inherited from rpyc.core.service.Service:
    |
    | __init__(self, conn)
    |       Initialize self.  See help(type(self)) for accurate signature.
    |
    | on_connect(self)
```

```
|       called when the connection is established
|
| on_disconnect(self)
|       called when the connection had already terminated for cleanup
|       (must not perform any IO on the connection)
|
| ----------------------------------------------------------------------
| Class methods inherited from rpyc.core.service.Service:
|
| exposed_get_service_aliases = get_service_aliases() from builtins.type
|       returns a list of the aliases of this service
|
| exposed_get_service_name = get_service_name() from builtins.type
|       returns the canonical name of the service (which is its first
|       alias)
|
| get_service_aliases() from builtins.type
|       returns a list of the aliases of this service
|
| get_service_name() from builtins.type
|       returns the canonical name of the service (which is its first
|       alias)
|
| ----------------------------------------------------------------------
| Data and other attributes inherited from rpyc.core.service.Service:
|
| ALIASES = ()
```

# REFERENCIAS

# Anexos

## Anexo A: Hardware (diagrama de construcción del robot, componentes principales)

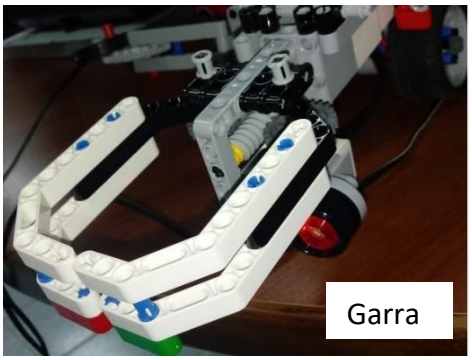
**Componentes principales**

El robot consiste de un motor mediano, además de tres motores grandes, un brick y una garra.

BRICK


Motor grande

Motor mediano



# Anexo B: Software (código de los programas implementados)

main.py

```
import kivy
kivy.require('1.9.0')

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout

class contenedor1(BoxLayout):
        None


class MainApp(App):
        tittle ="control Arturitoe"
        def build(self):
        return contenedor1()


if __name__=='__main__':
        MainApp().run()
```

nuv.py

```
import rpyc
from tkinter import *
from tkinter.ttk import *

window = Tk()


window.title("---ARTURITOE---")

window.geometry('350x200')

def frenar():
        conn.root.Frenar()
def avanzar():
        conn.root.Avanzar()

btn = Button(window, text="avanzar", command=avanzar)
btn_2 = Button(window, text="retroceder", command=retroceder)
btn_3 = Button(window, text="izquierda", command=izquierda)
```

```
btn_4 = Button(window, text="derecha", command=derecha)
btn_5 = Button(window, text="frenar", command=frenar)
btn_abrir = Button(window, text="abrir", command=Abrir)
btn_cerrar = Button(window, text="cerrar", command=Cerrar)
btn_girar = Button(window, text="girar", command=girar)

btn.grid(column=1, row=0)
btn_2.grid(column=1, row=1)
btn_3.grid(column=0, row=1)
btn_4.grid(column=2, row=1)
btn_5.grid(column=1, row=2)
btn_abrir.grid(column=0, row=0)
btn_cerrar.grid(column=2, row=0)
btn_girar.grid(column=3, row=0)

window.mainloop()
```

prueba2.py

```
#!/usr/bin/env python3
import rpyc
from rpyc.utils.server import ThreadedServer
from ev3dev.ev3 import *

class Robot(rpyc.Service):
    nombre="arturitoe"
    giro=True
    garra_abierta=True
    rueda_izq=Motor(OUTPUT_A)
    rueda_der=Motor(OUTPUT_D)
    garras=MediumMotor(OUTPUT_B)
    girar_garra=Motor(OUTPUT_C)



    def exposed_Abrir_Cerrar(self):

            if self.garra_abierta==True:
                    self.garras.run_timed(time_sp=800,speed_sp=1080)
                    self.garra_abierta=False
            else:
                    self.garras.run_timed(time_sp=800,speed_sp=-1080)
                    self.garra_abierta=True
```

```
        def exposed_Giro_Garra(self):

         if self.giro==True:
                self.girar_garra.run_timed(time_sp=500,speed_sp=400)
                self.giro=False
         else:
                self.girar_garra.run_timed(time_sp=500,speed_sp=-400)
                self.giro=True

        def exposed_speak_message(self, msg):
        Sound.speak(msg)


        def exposed_Avanzar(self):
         self.rueda_der.run_forever(speed_sp=500)
         self.rueda_izq.run_forever(speed_sp=500)

        def exposed_Retroceder(self):
         self.rueda_der.run_forever(speed_sp=-500)
         self.rueda_izq.run_forever(speed_sp=-500)

        def exposed_Girar_der(self):
        self.rueda_der.run_forever(speed_sp=200)
        def exposed_Girar_izq(self):
        self.rueda_izq.run_forever(speed_sp=200)

        def exposed_Frenar(self):
        self.rueda_der.stop()
        self.rueda_izq.stop()



if __name__ == "__main__":
        s = ThreadedServer(Robot, port=65535)
        s.start()
        print("Server up")
```

Anexo C: Comunicaciones (configuración de comunicación pc-robot considerando RPyC)

```
{
   "ev3devBrowser.additionalDevices": [
      {
         "name": "arturitoe",
         "ipAddress": "192.168.70.166"

      }
   ],

   "launch": {
      "version": "0.2.0",
      "configurations": [
         {
            "name": "Download and Run",
            "type": "ev3devBrowser",
            "request": "launch",
            "program": "/home/robot/${workspaceRootFolderName}/${relativeFile}"
         }
      ]
   },
   "ev3devBrowser.password" : "arturitoe"
}
```

---

[1] https://realpython.com/documenting-python-code/#documenting-your-python-code-base-using-docstrings

[2] https://python-para-impacientes.blogspot.com/2014/02/docstrings.html