- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video output
- CSI camera interface
- DSI display interface
- Micro SD card slot with pre-installed raspbian

The easiest way to work with your Raspberry Pi by is connecting a mouse, keyboard and a monitor and using it like a regular computer. This however only allows one person to work on it. In order to work more efficiently, other people from your team have the option to connect to your Raspberry Pi remotely using the SSH protocol. This protocol is best suited for console operation and is readily available on Linux and MacOS machines. Keep in mind that the 3pi Robot and the Raspberry Pi can be separated if this makes your team more efficient.

To connect when using Linux or MacOS, you can simply type

```
ssh -Y pi@a.b.c.d
```

in a shell window, replacing a.b.c.d with the IP address of the Raspberry Pi you want to connect to. You will then be asked for the password ("f2s").

When using Windows, you can choose different SSH clients but putty (https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe) is the one we would recommend. After downloading the executable, simply run it, enter the login credentials and save the session:

- Host Name (or IP address) = Raspberry Pi IP address
- Connection type = SSH
- Saved Sessions → enter a name, e.g. "f2s pi"
- Click "Save"
- From now on, double-clicking on the "f2s pi" entry in the list will automatically connect to the Raspberry Pi

3.2. Virtual Machine for 3pi and maze processing development (optional)

First, set up the virtual machine:

- Copy the virtual machine appliance files from the storage media that we pass around to any location on your computer
- Download VirtualBox from https://www.virtualbox.org/wiki/Downloads or use the installer from the storage media
- Install VirtualBox
- Run VirtualBox and import the virtual machine appliance
- Start the appliance

Once the virtual machine has booted, you'll be greeted by a Kubuntu Linux desktop. You don't really need to adjust anything if you don't really want to, it's sufficient to simply start the Eclipse IDE by clicking on the KDE menu, navigating to Applications and then Development.

With Eclipse started, you can find two projects in the project explorer on the left. The project named **3pi** is the skeleton project you should use to work on the 3pi Robot code while the **maze_processing** project provides you with a working setup of OpenCV to create the program that turns the maze image into a maze description.

both the raw and annotated images to disk.

- After the object detection has been performed, the module responds with a string of the format "**%s %f %f**", where
  1. %s represents the object found: "**nothing**", "**stop**" or "**rb**"
  2. %f represents the **confidence in percent** with which the object was identified
  3. %f represents the **distance in mm** of the detected object
- If more than one object is recognized in the frame, only the object with the highest confidence level is returned.
- After sending the response, the module waits for the next request.

Note: Please keep in mind that our module is good but not fool-proof. It is possible that not all road blocks will be recognized and the penalty will apply.

Last updated 2019-05-22 13:49:48 MESZ

```
#!/usr/bin/env python

import socket

#connection to SERVER
TCP_IP = 'localhost'
TCP_PORT = 5003
BUFFER_SIZE = 1024
socket_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

socket_server.bind((TCP_IP, TCP_PORT))
socket_server.listen(1)
print "listening on Port " + str(TCP_PORT)
conn, addr = socket_server.accept()

try:
    while 1:
        data = conn.recv(BUFFER_SIZE)
        recv_msg = data.decode() # The encoded message needs to be decoded to a string again
        print(recv_msg) #prints "Hello!"
        trans_msg = "Hello back!"
        conn.send(trans_msg.encode())
finally:
    conn.close()
```

3.6. Universal Asynchronous Receiver-Transmitter (UART)

The communication between the Raspberry Pi and the 3pi Robot uses the UART interface. While this might already be familiar to you as it's the most common communication interface in the embedded world, we'd like to give you a few hints on how to use it.

On the 3pi Robot side, it's easiest to use blocking reads when waiting for input data. You can use code like this to setup the communication using the serial functions declared in `lib/OrangutanSerial/OrangutanSerial.h`:

```
char buffer;

serial_set_baud_rate(115200);
serial_set_mode(SERIAL_CHECK);  // Don't use ISRs and background buffers
serial_receive(&buffer, 1);     // Configure ring buffer to use for receiving

while (1) {
  serial_check();  // Check if the UART hardware has received data and place it in the receive buffer
  if (serial_get_received_bytes() > 0) {
    print_character(buffer);  // Display received character on the LCD
  }
}
```

Note that this code only works when the sender sends single bytes as the ring buffer is one byte long. If you want to send more than one byte at once then you'll need to increase the ring buffer size and handle it. The supplied `slave.cpp` program can serve as an example if you want to do this, and keep in mind that there's the library reference available as well. Either way, sending out data is even easier than receiving:

```
serial_send_blocking("Hello World");
```

Using the Kubuntu VM, you can test this connection using miniterm.py:

```
miniterm.py /dev/ttyUSB0 115200
```

or, if you need to send binary data and want to enter it using hexadecimal digits:

```
miniterm.py --encoding hexlify /dev/ttyUSB0 115200
```