Chapter 4

Preparation for the Radar Lab

The radar lab exercise will demonstrate how the Doppler shift of electromagnetic waves can be used to estimate a moving object's velocity relative to a sensor (the radar). The radar can optionally also be used to estimate the distance to an object by modulating the radar frequency (FMCW-mode), but is not a required part of the exercise.

4.1 Connecting the Radar Module to the Core System

We will use the K-LC6 v2 24 GHz radar module from RFbeam. It is a small radar with two rows (arrays) of eight antenna elements, one row for transmitting and one row receiving the electromagnetic waves. The antenna elements are laid out 1 x 8 elements, which gives us a narrow beam in the azimuth direction and a broad beam in the elevation direction. Read the datasheet and try to figure out which plane has the narrow beam (i.e. what is the azimuth and elevation directions for the radar). All electronics necessary to transmit and receive signals are included onboard the module.

4.1.1 Radar Module Layout

Figure 4.1 shows the pin layout of K-LC6 along with physical dimensions and what signals should be connected to each pin. **Take care not to connect anything wrong!** The radar modules are relatively expensive, and we don't have many more than we need.

The radar module has an input called $\mathrm{VCO_{in}}$ (Voltage Controlled Oscillator) which can be used to modulate the frequency of the radar wave (almost) linearly with the voltage on the $\mathrm{VCO_{in}}$. For the Doppler radar operation, we will simply leave this pin open (not in use), and the radar will default to a stable frequency at about 24.13 GHz.

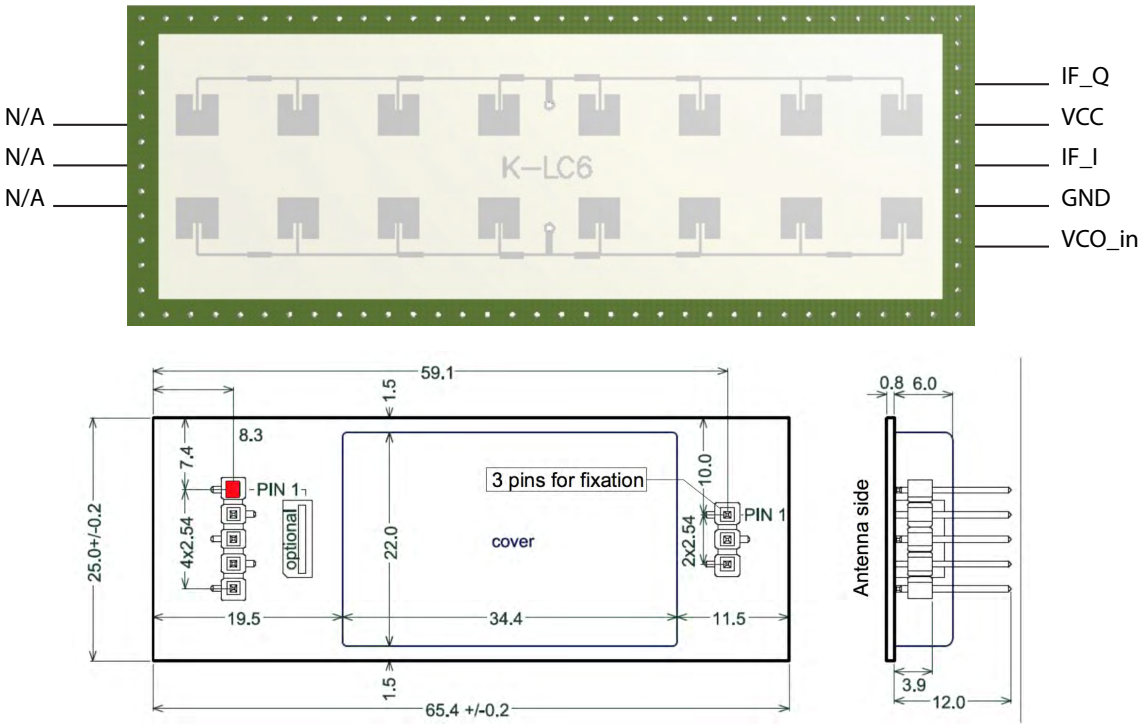
The K-LC6 needs 5V DC to operate. We recommend to use the 5V output from a Digilent kit to supply the radar since the 5V output from Rpi is not stable or powerful enough to drive the radar.

The two outputs $\mathrm{IF_I}$ and $\mathrm{IF_Q}$ are so-called *in-phase* (I) and *quadrature* (Q) signals. This means that the quadrature signal is phase shifted $\pm 90°$ relative to the in-phase signal (sign depends on direction of movement). In-phase and quadrature signals are very much used in electronic measurement systems. Check out the Wikipedia entry for initial information if you like.[1] From the radar module the in-phase and qudrature signals will be identical, though, only phase shifted to give the direction of movement of an object in range.

4.1.2 Active Bandpass Filters for the Radar Outputs

Due to relatively small reflected signals from typical objects, we need to amplify the output I- and Q- signals from the radar module. The bandpass filter should have a low-frequency cut-off of 3.5 Hz to remove any DC-offset from the I- and Q- signals, and a high-frequency cut-off of 4800 Hz to

[1] For example, you can find the real and imaginary values of an unknown impedance by measuring the in-phase and quadrature amplitudes of a known AC signal applied to the unknown impedance.



Pin	Description	
1	IF output Q	
2	VCC	5V supply
3	IF output I	
4	GND	0V supply
5	VCO in	1V .. 10V, open = 5V

Figure 4.1: Radar module K-LC6 pin-layout, physical dimensions and signal ranges for the pins. Take care not to connect it the wrong way, as doing so will reverse the applied voltage and potentially burn the internal power circuit. The figures are compiled from the datasheet and modified for easier understanding, and are thus courtesy of RfBeam.

suppress high frequencies that are outside the range of interest. In addition, since the radar module needs 5 V and our ADC runs off 3.3 V, we need to lower the DC offset of the I- and Q-signals from 2.5 V to 1.67 V to match wit the ADC inputs.

We will use an active bandpass filter for this, built around the operational amplifier MCP6002. This IC has two op-amps inside, so we will build two active bandpass filters, one for each radar channel. Make sure you look into the MCP6002 datasheet so that you know your device, and particularly the pin layout!

In the lab kit, you should find all necessary components for the active filters except the resistors needed to define the new reference (DC offset). Find some suitable ones from your inventory. Figure 4.2 shows a sketch of the schematic for one of those active filter circuits.

4.1.3 Wiring up and testing the Active Bandpass Filters

Take time to plan a bit before you start to wire up the active band pass filters. There are quite a few components, so the breadboard area gets crowded quickly. Read the MCP6002 datasheet carefully so that you connect things correctly. Check the wiring two or three times before you connect the power. Draw your own schematic based on Figure 4.2, and have a look at Figure 4.3 for inspiration.

When you have the filters ready you should test them with a signal generator and oscilloscope, to verify that they function properly *before* you connect the radar module and ADCs. **This is essential documentation for your project report.**

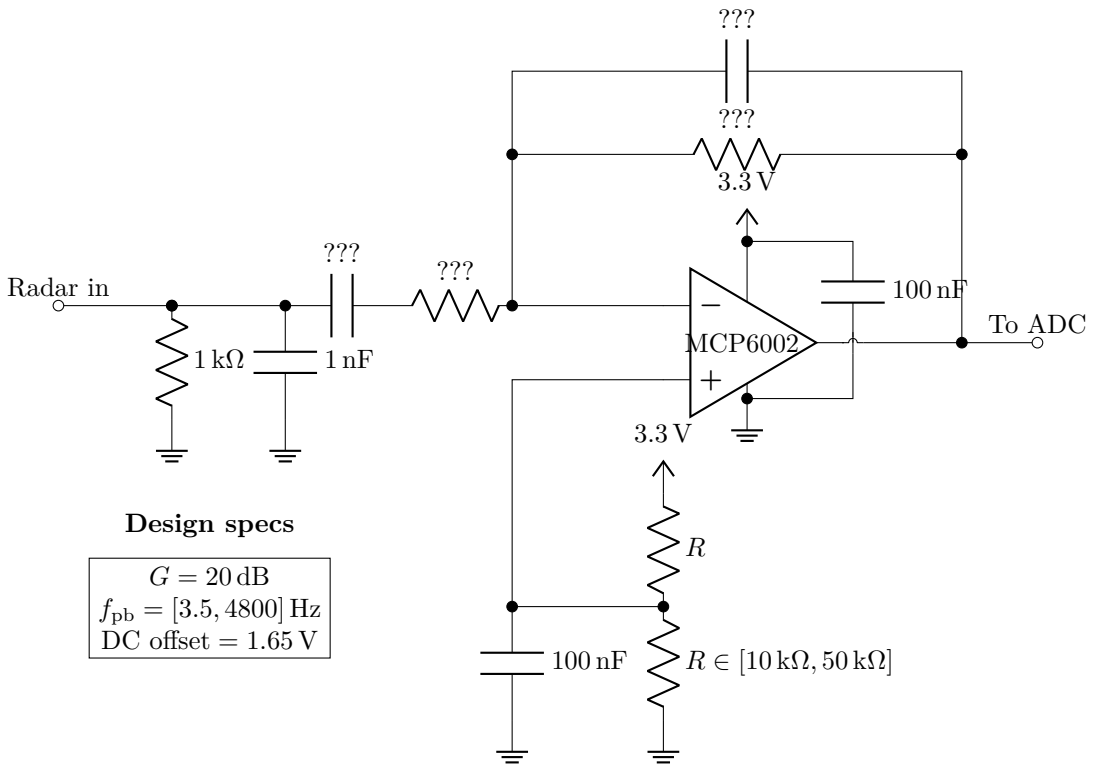


Figure 4.2: Circuit diagram of an active bandpass filter with DC offset. The in-phase or quadrature signal is used as input. The op-amp is a single supply IC with rail to rail output swing, i.e the output signal will always be in range $u \in [0, 3.3]$ V, equivalent to our ADCs' measurement range.

Use the Digilent kit to measure the frequency response of the active bandpass filters. The gain should be 20 dB in the pass band, and the lower and upper cut-off frequencies should be close to the specified values. In addition, you must check that the DC voltage offsets on each side of the circuits are as expected.

If your filter response looks ok, everything should be ready for connecting the radar module to the input and the ADCs to the output.

4.1.4 Wiring up the Radar Module

The radar module has 3 pins on the left side that are not connected to anything and merely used for support/fixation. We will not use those at all. Instead, we will use a set of headers with a resulting 90° angle to get the radar module to stand upright.

Route the 5 V and GND signals to pins 2 and 4, respectively. Use an external 5V supply from a Digilent kit instead of the 5V output from the Rpi.

Continue with connecting the in-phase output signal (pin 3), IF_I, and the quadrature output signal (pin 1), IF_Q, to each channel of the active bandpass filters. Then connect the filter outputs to ADC channels 4 and 5. See Figure 4.3 for inspiration. Take a little time to think whether the radar now has the narrowest beamwidth in the vertical or in the horizontal plane.

When you have triple-checked that your wiring is correct, mount the radar module carefully into the breadboard with the 90° header mount. Make sure you hit the right lines, and that you have it turned the correct way (we don't want to reverse bias it!).

The mounted radar module should look something like Figure 4.4, where you can see the complete system with the microphones and radar together, connected to the ADCs.

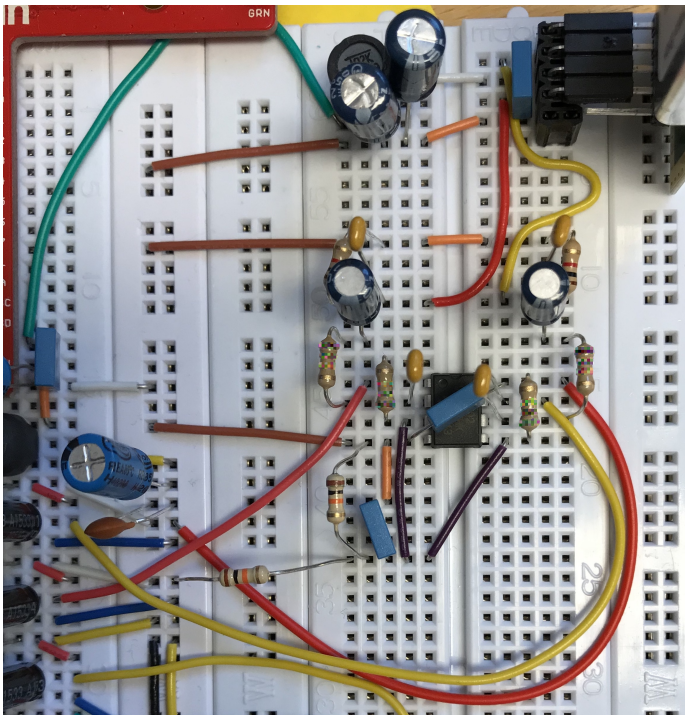


Figure 4.3: Example of two parallel active band pass filters, one for each radar signal channel.

4.2 Testing the Radar Module

You are now ready to test the radar. With an oscilloscope connected to the ADC inputs in parallel, you can see the signals live by moving a hand or book or something in front of the radar. Note that the Doppler shift will only be produced when the object is moved radially towards or away from the radar.

Also do some test runs with the sampling program. Try to move the object with a stable velocity away from or towards the radar module while sampling. Look at the data in Python. You should detect two sinusoidal signals that are phase shifted ±90° depending on the direction you move the item. Next step is to write Python code for FFT to analyze the Doppler shift in the frequency domain. Remember to use a complex FFT based on combining the $\mathrm{IF_I}$ and the quadrature output signal, $\mathrm{IF_Q}$, into a a complex signal.

$$S(f) = FFT(IF_\mathrm{I} + \mathrm{j}IF_\mathrm{Q}) \tag{4.1}$$

When using a complex signal, the spectrum should be asymmetric showing a peak in the spectrum either at the positive or negative Doppler shift.

Hint: Try to multiply the raw signal with different window functions (Hanning, Hamming, Kaiser etc) before taking the FFT to study how they will affect the sidelobes of the spectrum. This is best viewed when plotting the spectrum using dB. It is also important to remove the DC-offset before taking the FFT. (Try with and without removing the DC-offset to see the difference.)

The frequency of the recorded signal can be used to estimate the velocity according to the preparatory exercises.

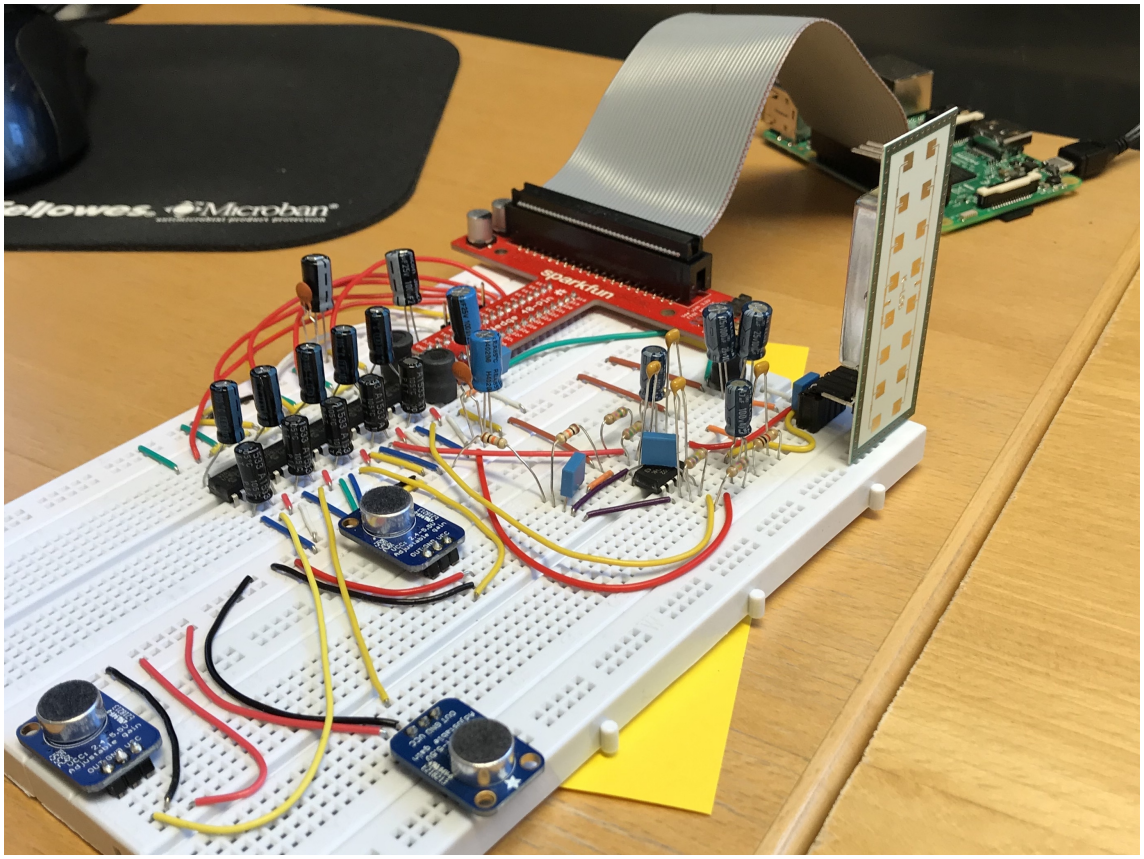


Figure 4.4: Complete system after lab sessions 1, 2 and 4. You can now locate the direction of a sound source, and its radial velocity if it is within the radar's angular range, at the same time.

Chapter 5

Preparation for the Optics Lab

5.1 Introduction

The Optical Lab exercise will demonstrate how we can use an ordinary low-end consumer camera to detect a person's heart rate, both by reflected light from the skin and via transmitted light through a finger or similar. For descriptions on the lab exercise, refer to the lab text. Below, we will go through the steps necessary to the get the camera up and running.

5.2 Raspberry Pi Camera Setup

5.2.1 Connecting to the Camera Serial Interface (CSI)

The RPi has a special contact for the picamera:

1. Pull the white handle carefully up and towards the Ethernet port.
2. Insert the ribbon cable with the blue side facing the Ethernet port.
3. Push the white handle carefully down.

See figure 5.1. See also the very first lab lecture done at the beginning of the course, uploaded to BlackBoard, and https://www.raspberrypi.org/documentation/configuration/camera.md.

5.2.2 Software setup

The instructions in this box concern the Buster version of Raspbian. The legacy camera module is not available on Bookworm.
After having connected the camera to the RPi board, first enable the camera by running `sudo raspi-config`, and then selecting 'Interface Options', 'Camera' and choose to enable it. It might also be that you have to add yourself to the "video" group in order to give your user enough privileges to access the video input, which is done by running `sudo gpasswd -a [username] video`, where you insert the username of your main user on the system instead of '[username]'. This is probably not necessary if you still use the default user "pi". During camera enabling, raspi-config edits a config file which is read only during boot, so we have to reboot. Reboot the RPi, e.g. by typing `sudo reboot`. After reboot, the camera should be ready.

Starting with Bullseye, the Raspberry Pi camera is already activated. Verify that the camera is accessible by executing

```
rpicam-hello
```

on the Pi (requires `ssh -Y`). If you `ssh`-ed to the Pi without the `-Y` flag, you can test camera functionality by executing

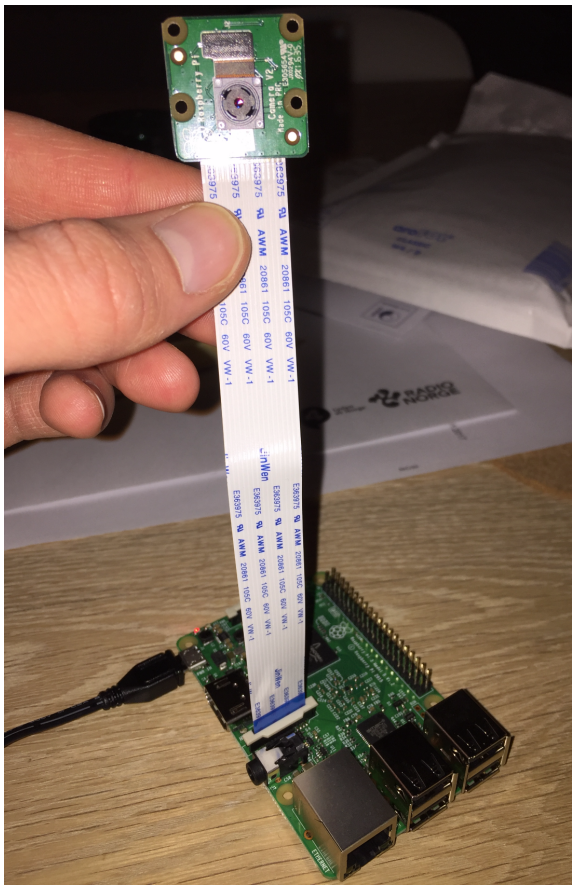


Figure 5.1: The picamera hooked up to an RPi.

```
rpicam-still -v -o foo.jpg
rpicam-vid -v -o foo.h264
```

instead. Note that `sudo` privileges may be required.

The `raspistill` and `raspivid` applications are used on Buster. They are legacy and will not work on later versions of the OS.

Consult this page for more details on `rpicam` applications.

Verify that there were no errors, and that any produced images or videos make sense. In order to properly play the video, see section 5.4.2.

A couple of software libraries are necessary for the video acquisition script that you will use later. The default installation should already have these `apt` packages installed, but if not, you can install them using

```
sudo apt install python-picamera2 python3-picamera2 gpac
```

`python-picamera2` is the Python module for accessing the camera, and `gpac` is a package which includes a tool for wrapping video files in MP4 containers. If `apt` is not found, try `apt-get`.

5.3 Preparations for the Lab

We have provided you with a Python script that takes care of the video recording and conversion of the video file to a signal that can be processed.

`record_video.py` Record a video.

`read_video_from_roi.py` Choose a spatial slice of the video, and find the average of each colour channel in the region of interest for each frame.

The Python video acquisition script must be accessible from the Pi.

Read the comments in the video acquisition script to see what is being done there and instructions for how to run it. Identify parameters that might need to be changed during acquisition.

5.3.1 Some notes

- Make sure that the region being measured has sufficient amount of light, and also that the light does not cause specular reflection and white spots in the video.
- Changing the ISO value has caused some problems after the h264 is converted into mp4, so it is recommended using the default `iso=10` from the Python script and rather change the `awb_gains` value. You might need different `awb_gains` values for different points you want to measure on.

5.4 Information about the video recording scripts

5.4.1 The Picamera Library

It is possible to access and use the camera in different ways, but for this lab we recommend that you use the `picamera2` Python library. The library allows us to control the camera with the simplicity and readability that Python offers. If you want to experiment a bit yourself – and we recommend this approach – have a look at the `picamera2` documentation here. Here you will find basic examples explaining how to capture regular images and regular videos, along with some more advanced examples when you feel ready for it.

In addition to checking out some examples to familiarize yourselves with the library we recommend you to check out this. Here we can see the supported resolutions we can choose from, and at what frame rates we can capture video in at the different resolutions.

5.4.2 Video Codec

A video codec is a device or software which has the purpose of compressing or decompressing a digital media file. We will use a codec called *h264*. This codec does not usually contain information about frame rate, length and such. If you try to play the .h264 file directly in VLC or a another video player, notice that it will not be able to display the length of the video, and depending on the frame rate chosen for the recording in your Python program it will probably be played at a faster or slower rate than what you expected. The reason for this is that it cannot find information about the frame rate in the .h264 file, so it will play it using a default frame rate. Thus, we need to wrap it in a container so that we can process the information afterwards. A container can contain different type of media files. If you have a video file encoded with the h264 standard, an audio mp3 file and a srt subtitle file, we can put all of these in a container so that we have a video with audio and subtitles. The container format we will use is the `mp4` format.

We use the pre-installed FFmpeg library to do the conversion from a file `input.h264` captured at a frame rate 40 to *output.mp4*:

```
ffmpeg -framerate 40 -i input.h264 -c copy output.mp4
```

This is already done in the example Python file we have provided for you, called **record_video.py** on Blackboard.

Now, if you try opening the new mp4 file in VLC again you will see that we have information about the time, and the frame rate it is played at should be correct.

5.5 On-Board Processing (optional)

After you are finished with the implementation of your pulse detection algorithm, have acquired the required data, and you are well satisfied with the results, you might also want to try implementing your algorithms on-board the RPi and try to estimate the pulse in real-time, or just in Python on the RPi.

See the acoustic lab text for general tips for processing in Python. We use the same techniques in the optical lab: cross-correlation and band pass filtering. Finding maxima might be a bit more difficult. Here, you could use something called smoothing splines to fit the noisy autocorrelation using smooth third degree polynomials, and use built-in functionality for estimating the derivatives and the roots of the derivatives.

For real-time processing, this will probably get a bit more involved. You might need to do this in a sliding window-way: Decide on a number of samples you need to collect in order to be able to estimate a pulse. Wait until you have collected this number of samples. Estimate the pulse. Then move your window one sample with the time direction, and continue until finish, possibly let the window skip a couple of more samples depending on the computational requirements of the processing.

PiCamera has support for writing to a circular ring buffer (http://picamera.readthedocs.io/en/release-1.12/api_streams.html). This could be used to process data in parallel with the video acquisition. This could be combined with OpenCV, which has image processing routines which could be used for object tracking, face detection, etc.