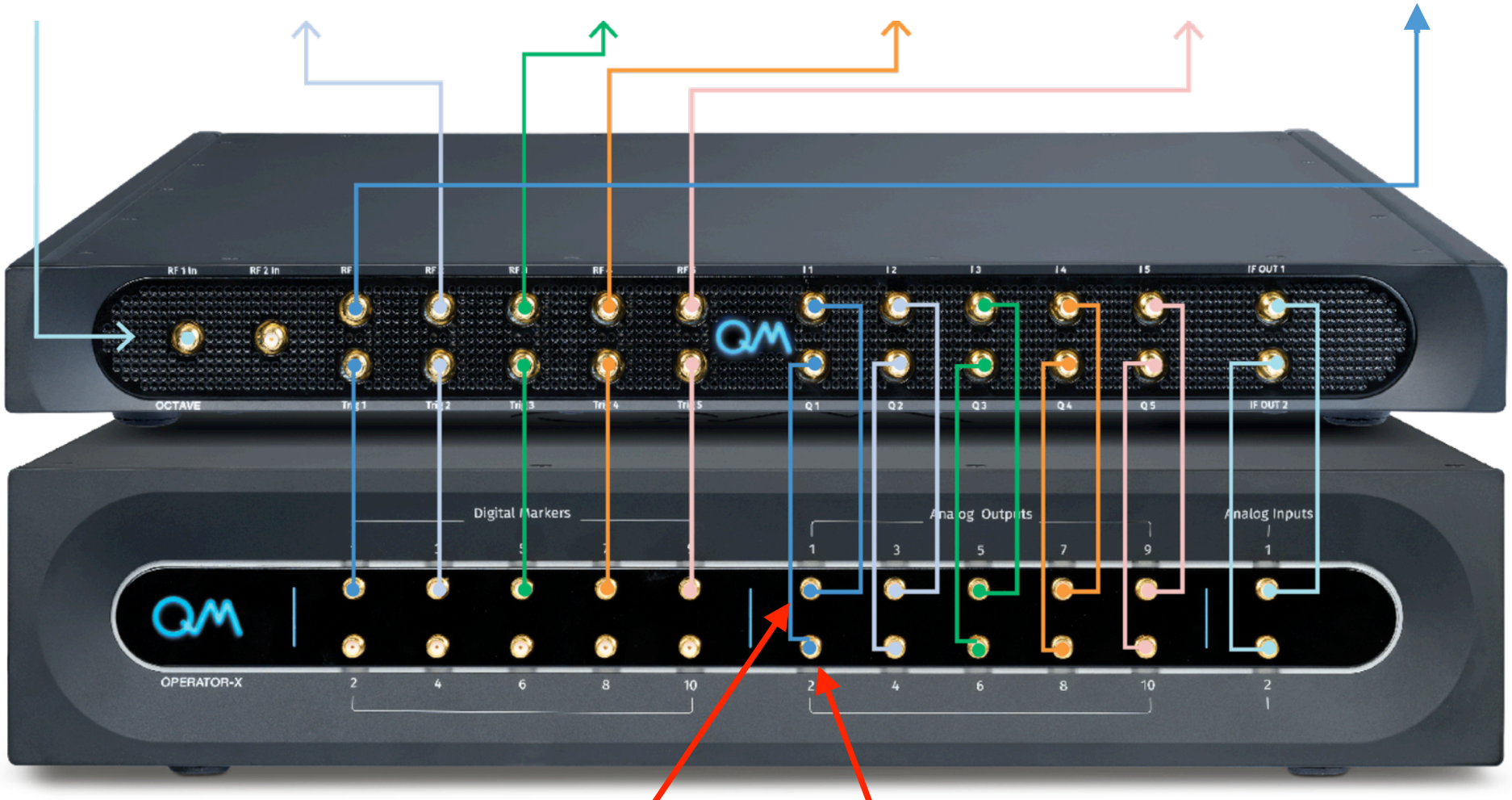
Contrôler et mesurer un qubit avec une Quantum Machine

- **Points de cours:**
  - Rotations arbitraires sur la sphère de Bloch
  - Modulation IQ
    - Modulation à Bande Latérale Unique (BLU)
- **Modulation IQ & BLU avec la Quantum Machine**
- **Point de cours:**
  - Démodulation IQ
- **Mesure dans le plan IQ avec la Quantum Machine**

Modulation IQ avec la Quantum Machine

$$I\cos(\omega_{\mathrm{LO}}+\omega_{\mathrm{IF}})t + Q\sin(\omega_{\mathrm{LO}}+\omega_{\mathrm{IF}})t$$



Octave

OPX

$$I\cos\omega_{\mathrm{IF}}t + Q\sin\omega_{\mathrm{IF}}t$$

$$-I\sin\omega_{\mathrm{IF}}t + Q\cos\omega_{\mathrm{IF}}t$$

Lien vers la doc: https://docs.quantum-machines.co/latest/

Démarrage

- Lancez Spyder, vérifiez que le kernel Python est en 3.12
- Choisissez Qt5 pour la sortie graphique dans les préférences, ou exécutez %matplotlib qt5 dans la console **avant** de lancer un des programmes
- Le répertoire de travail est \Desktop\TP Quantum Machine
- Le répertoire \TP Quantum Machine\tpqm sera régulièrement mis à jour, pour cela vous devrez faire un clic droit, git bash puis exécuter « git pull » dans la console
- Vous devez copier les fichiers de ce répertoire vers votre propre répertoire, ne travaillez pas directement dans \tpqm
- Vérifier dans un browser que vous voyez la QM à l'adresse 192.168.88.251 ou 192.168.88.253. Notez le nom du cluster, l'adresse et le nom du cluster doivent être copiés dans le fichier « qm_ip.py ».
- Depuis Spyder, exécutez configuration.py. Vous devriez voir un message de connection à la QM.
- **Pensez à régulièrement sauver des données, figures,…**

Planning des expériences

- Visualisation des signaux générés par l'OPX à ω_{IF}
- Visualisation des signaux générés par l'Octave à $\omega_{\mathrm{LO}} + \omega_{\mathrm{IF}}$
  Calibration des mixers
- Communication sans fil par modulation/démodulation dans le plan IQ
- Mesure de distance avec un radar multi-fréquences
- Mesure de vitesse avec un radar mono-fréquence
- Mesure de distance et de vitesse avec un radar multi-fréquences

Hello QUA

```
from qm.qua import *
from configuration import qm
```

Chargement de la configuration et ouverture de la QM

```
####################
# The QUA program #
####################
with program() as prog:
    with infinite_loop_():
        play("pulse", "rf1")
        wait(10*u.us)
```

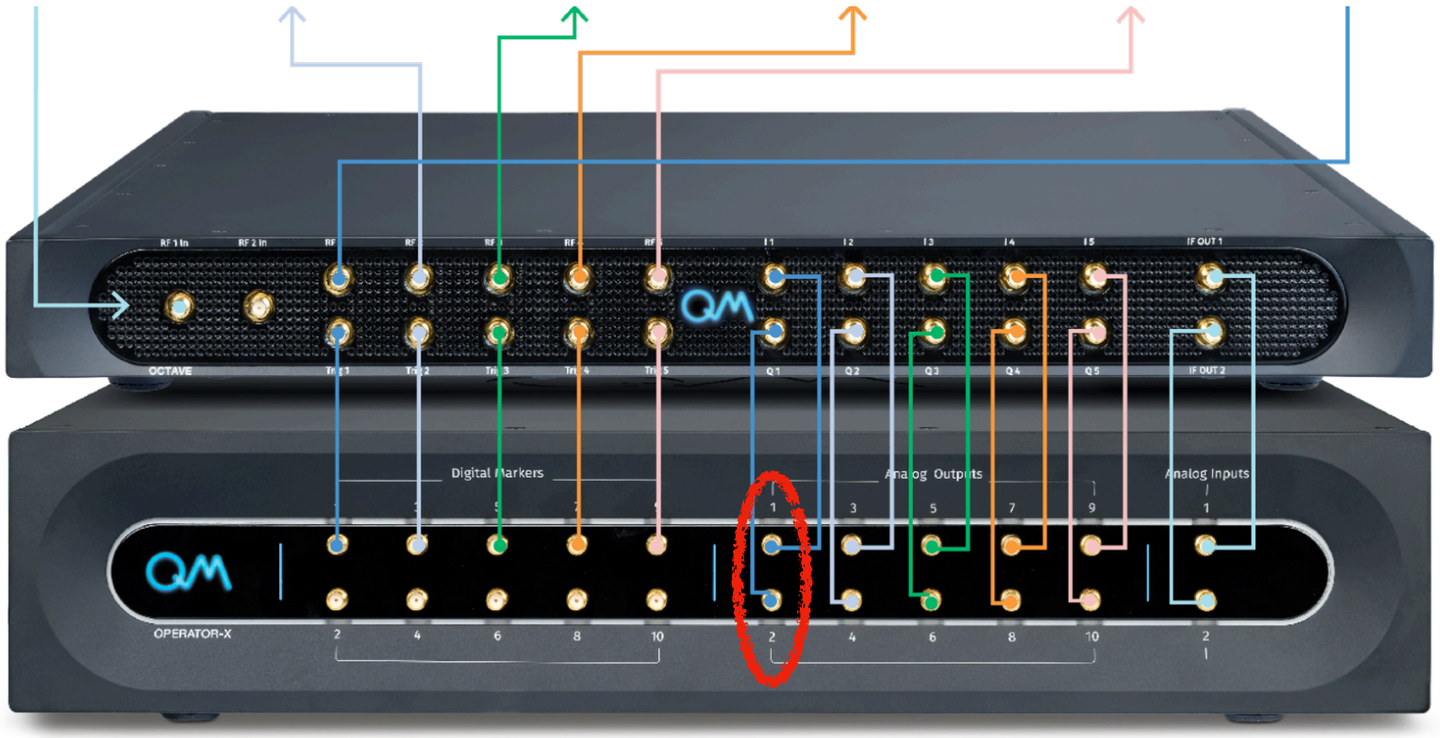
Définition du programme

```
######################
# Execute the program #
######################
job = qm.execute(prog)
```

Lance l’exécution du programme sur la QM

Pour arrêter le code: job.halt() dans la console

Visualiser les signaux de l’OPX

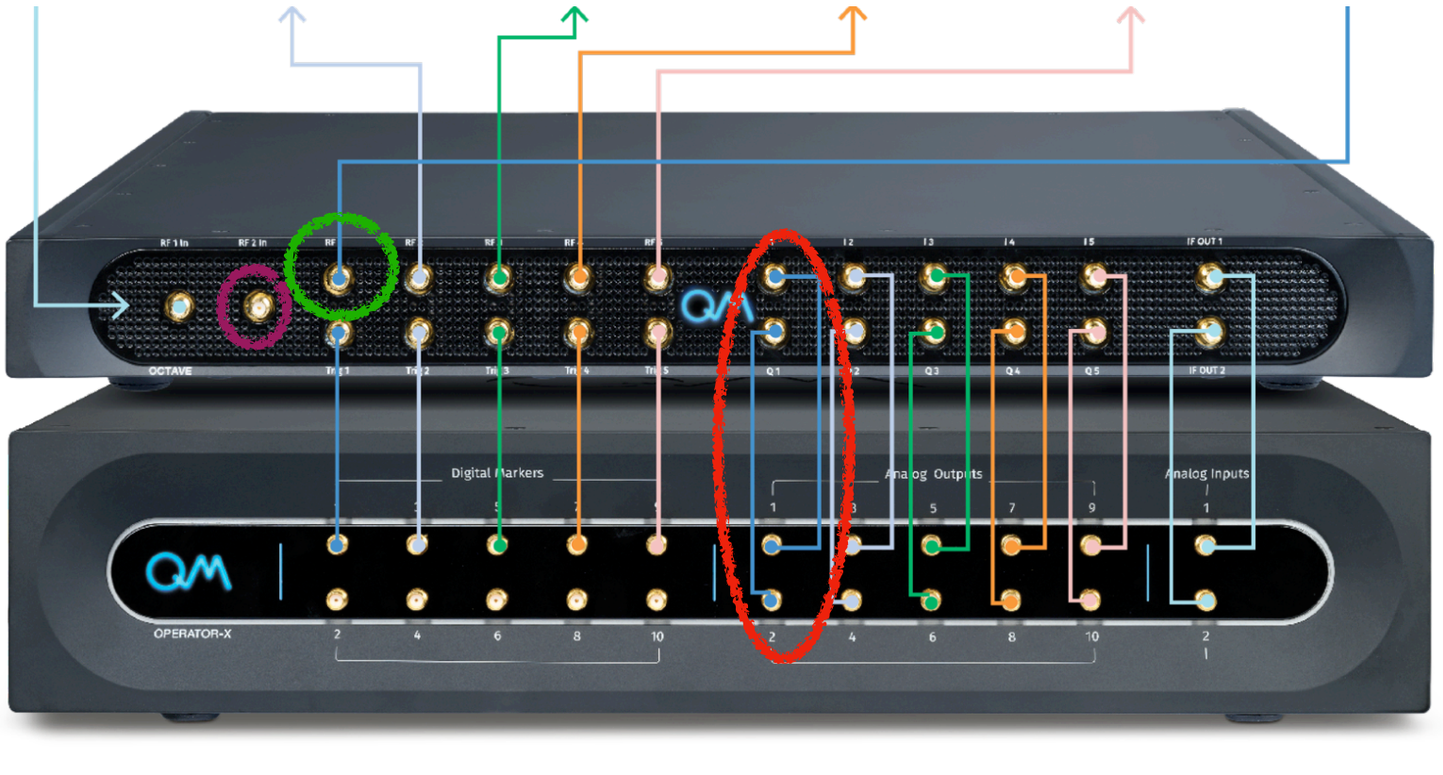


Octave

OPX

- Branchez les sorties analogiques 1&2 de l’OPX sur l’oscilloscope. Exécutez le programme « 01_play_pulse », modifiez la fréquence et l’amplitude du pulse en modifiant « configuration.py »
  - Quel signal est obtenu en supprimant le wait ?
  - Modifiez le programme pour obtenir deux pulses séparés de 40ns suivi d’une pause de 10µs
  - Mesurez et comprenez les phases entre les signaux de chaque pulse
  - Modifiez le programme pour obtenir deux pulses déphasés de π/2 en utilisant la commande amp() avec quatre paramètres.

Visualiser les signaux de l’Octave



Octave

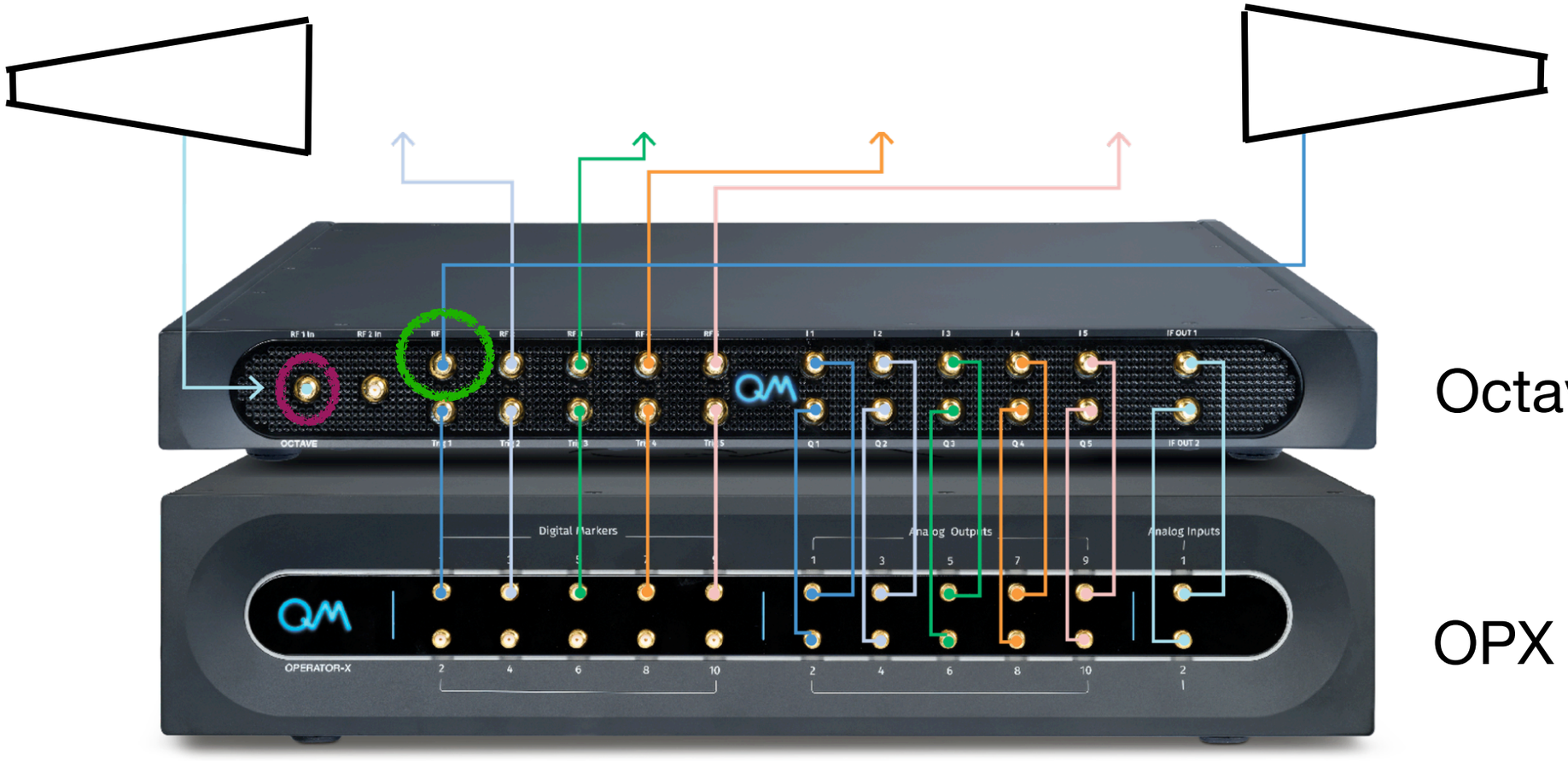
OPX

- Brancher les sorties analogiques 1&2 de l’OPX sur l’Octave
- Brancher la sortie RF1 sur RF2 in pour utiliser la QM comme un analyseur de spectre
- Exécutez le programme « 02_spectro ». Les paramètres de config sont dans « configuration_spectro.py »

Calibration des mixers IQ

- Evaluez les puissances relatives entre les différents signaux visibles sur le spectre
- Appliquez la procédure de calibration « calibrate_mixer.py »
- Observez à nouveau le spectre
- Changez les offsets DC sur les mixers en exécutant dans la console qm.set_output_dc_offset_by_element(‘rf1','Q',-0.01) et essayez avec d’autres valeurs
- Quand la machine est calibrée, évaluer l’atténuation des composantes parasites par rapport à la composante principale

Communication par modulation IQ



Octave

OPX

- Brancher la sortie RF1 sur une antenne pour émettre le signal et l'autre antenne sur RF1 in pour recevoir le signal
- Exécutez le programme « 03_modIQ ». Les paramètres de config sont dans « configuration_modIQ.py »

Démodulation avec la QM

```
with program() as prog:
    I = declare(fixed)  # QUA variable for the measured 'I' quadrature
    Q = declare(fixed)  # QUA variable for the measured 'Q' quadrature
    I_st = declare_stream()  # Stream for the 'I' quadrature
    Q_st = declare_stream()  # Stream for the 'Q' quadrature
```

```
    with infinite_loop_():
        play("pulse", "emitter")
```

Emission par modulation IQ

```
    with infinite_loop_():
        # Demodulate the signals to get the 'I' & 'Q' quadratures)
        measure(
            "readout",
            "receiver",
            None,
            dual_demod.full("cos", "sin", I),
            dual_demod.full("minus_sin", "cos", Q),
        )
        # Save the 'I' & 'Q' quadratures to their respective streams
        save(I, I_st)
        save(Q, Q_st)
```

Réception par démodulation IQ

```
with stream_processing():
    # Cast the data into a 1D vector, and store the results on the OPX processor
    I_st.buffer(n_points).zip(Q_st.buffer(n_points)).save("IQ")
```

Envoi des données au PC par paquets de n_points

Live plot des résultats

```
####################
# Live plot        #
####################
class myLivePlot(LivePlotWindow):
    def create_axes(self):
        # Create plot axes
        self.ax = self.canvas.figure.subplots()
        # Plot
        self.spectrum = self.ax.plot((spectro_LO + frequencies)/1e6,np.ones(len(frequencies)))[0]
        self.ax.set_xlabel('Frequency (MHz)')
        self.ax.set_ylabel('Signal (dB)')
        self.ax.set_ylim(-120,-10)
```

Création du plot *(Appelé une fois au début)*

```
    def polldata(self):
        # Fetch the raw ADC traces and convert them into Volts
        IQ = self.job.result_handles.get("IQ").fetch(1)
        if IQ is None:
            return
        I = IQ['value_0']
        Q = IQ['value_1']
        self.spectrum.set_ydata(10*np.log10(I**2+Q**2))
        self.canvas.draw()
```

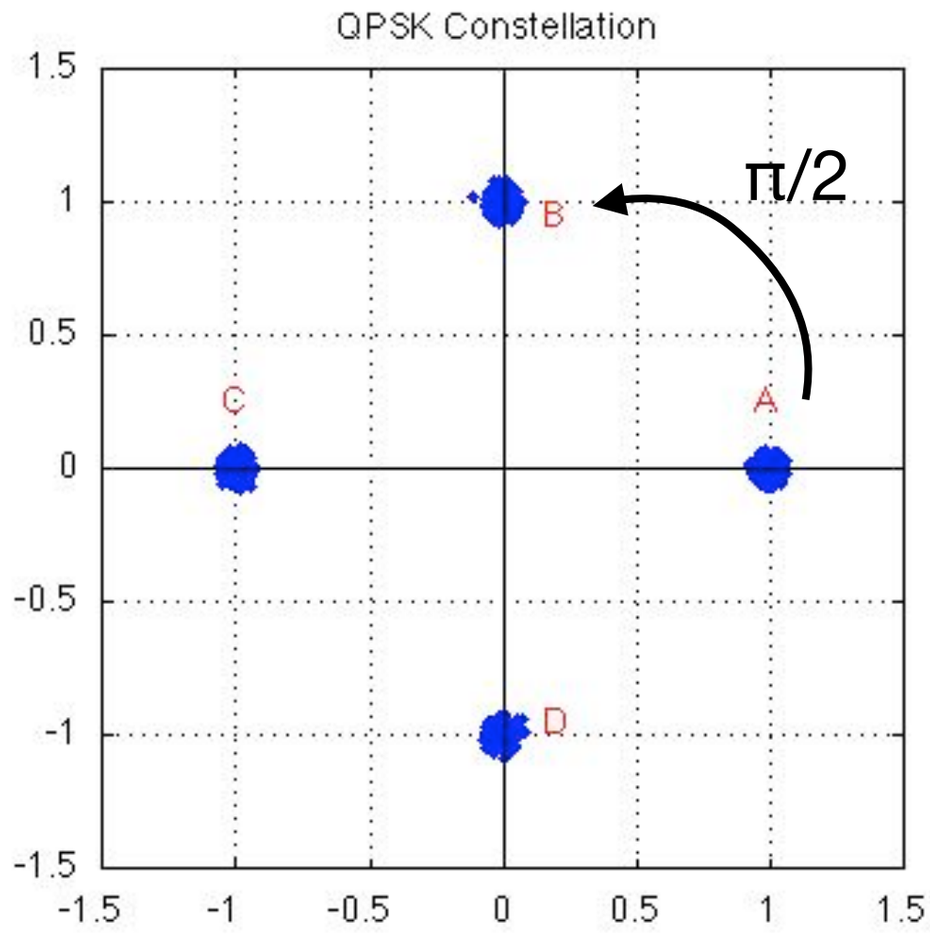
Récupération des données et mise à jour des données du plot

Cette fonction est exécutée tous les 100ms.

```
########################
# Execute the program #
########################
job = qm.execute(prog)
qm.set_io1_value(1.0)
window = myLivePlot(job)
window.show()
```

Communication par modulation IQ

- Modifiez 03_modIQ pour envoyer en continu des pulses de 77µs dont la phase tourne de π/2 à chaque pulse



- Démodulez pendant 10 µs et observer le résultat comme ci-dessus

Communication par modulation IQ

- Changez la durée des pulses à 20µs et utilisez le code suivant pour l'émission

```
msg=np.load("msg_1007.npz")
msgI=msg["x"]
msgQ=msg["y"]

with program() as prog:
    m = declare(int)
    Im = declare(fixed,value=msgI)
    Qm = declare(fixed,value=msgQ)
    I = declare(fixed)  # QUA variable for the measured 'I'
quadrature
    Q = declare(fixed)  # QUA variable for the measured 'Q'
quadrature
    I_st = declare_stream()  # Stream for the 'I' quadrature
    Q_st = declare_stream()  # Stream for the 'Q' quadrature

    with infinite_loop_():
        with for_(m, 0, m < len(msgI), m + 1):
            play("pulse" * amp(Im[m],0.,0.,Qm[m]), "emitter")
```

Changer la ligne 108 dans configuration_modIQ pour avoir une amplitude sur I et Q

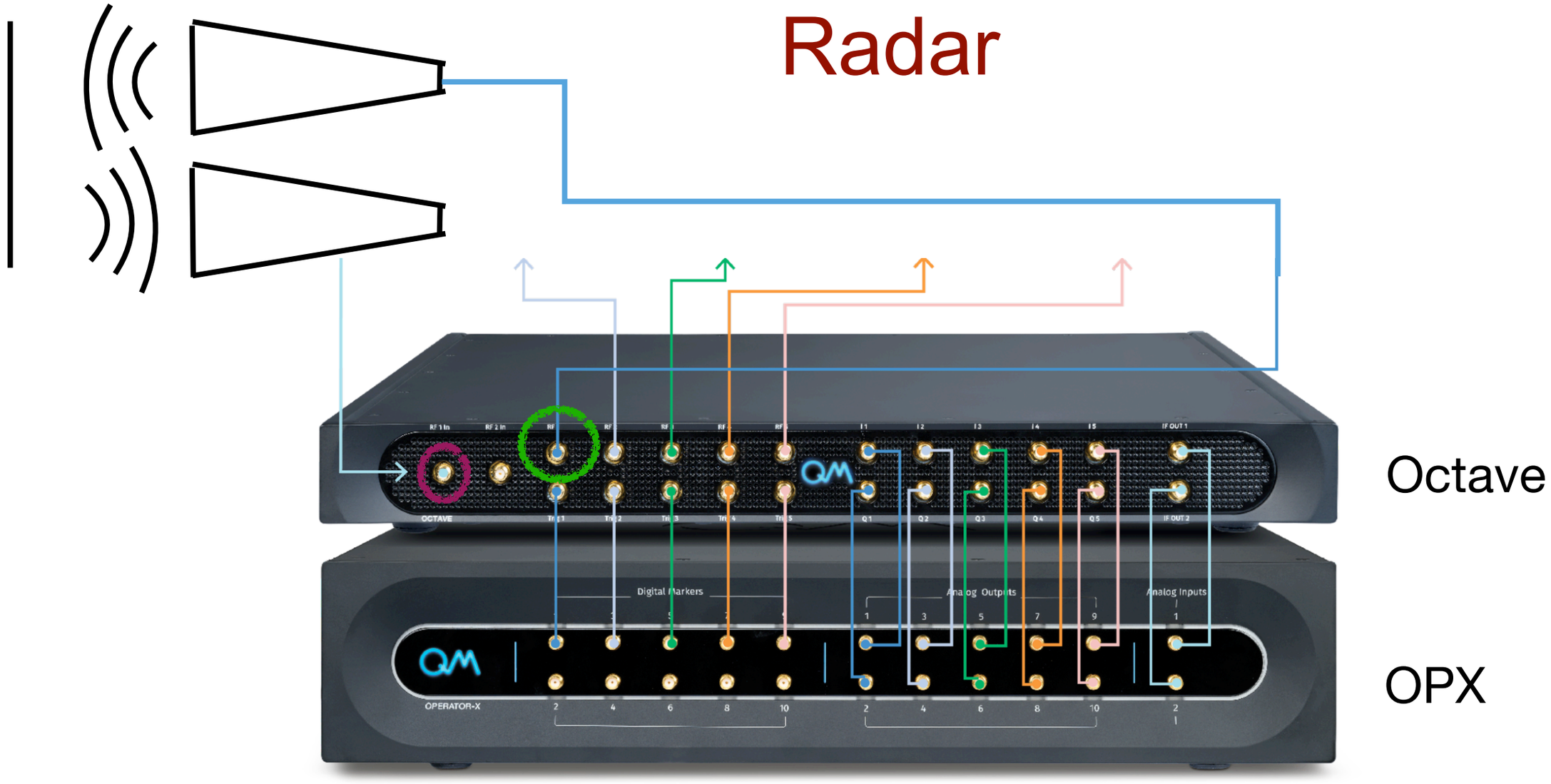
```
"pulse": {
    "operation": "control",
    "length": pulse_len,
    "waveforms": {
        "I": "const_wf",
        "Q": "const_wf",
    },
```

- Démodulez sur RF2 pendant 10 µs et observez le résultat

Communication entre deux machines différentes

- Refaites les expériences précédentes en envoyant un message à vos voisins et en réceptionnant leur message
- Essayer de trouver des astuces pour améliorer la transmission

Radar

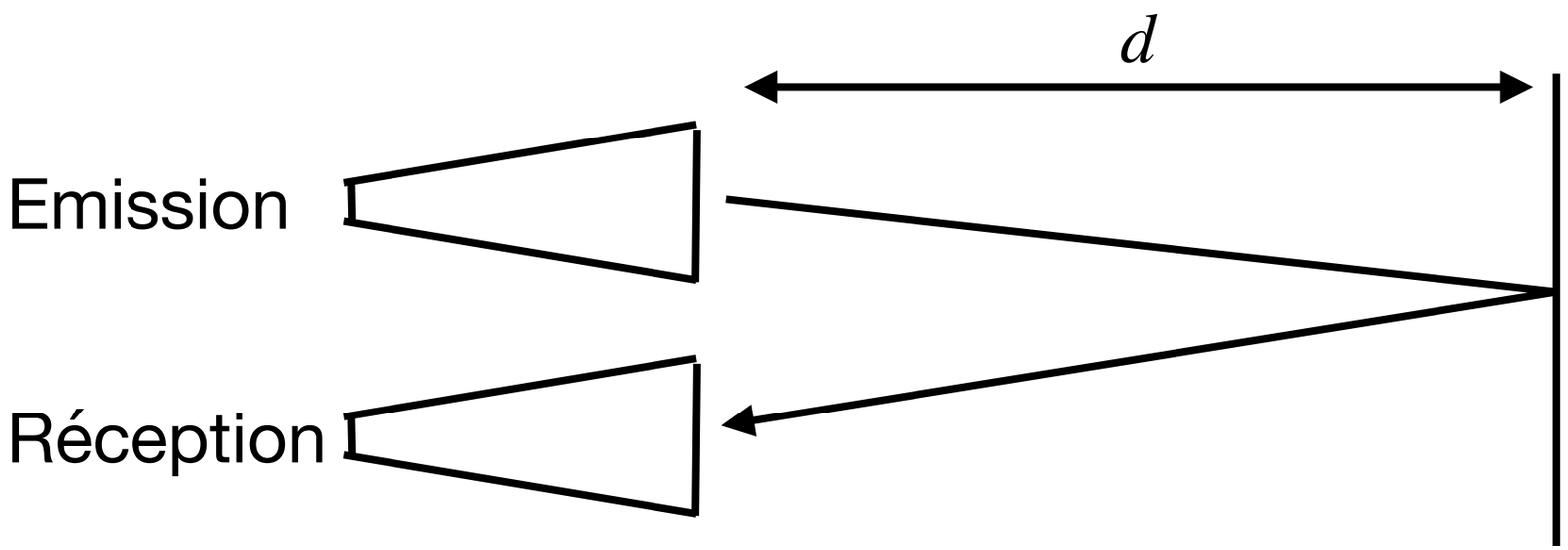


Octave

OPX

- Brancher la sortie RF1 sur une antenne pour émettre le signal et l'autre antenne sur RF1 in pour recevoir l'echo radar
- Exécutez le programme « 04_radar ». Les paramètres de config sont dans « configuration_radar.py »
- Analysez le programme et essayez de comprendre ce qui se passe quand vous bougez les antennes

Ranging



- Introduire le temps d'écho τ

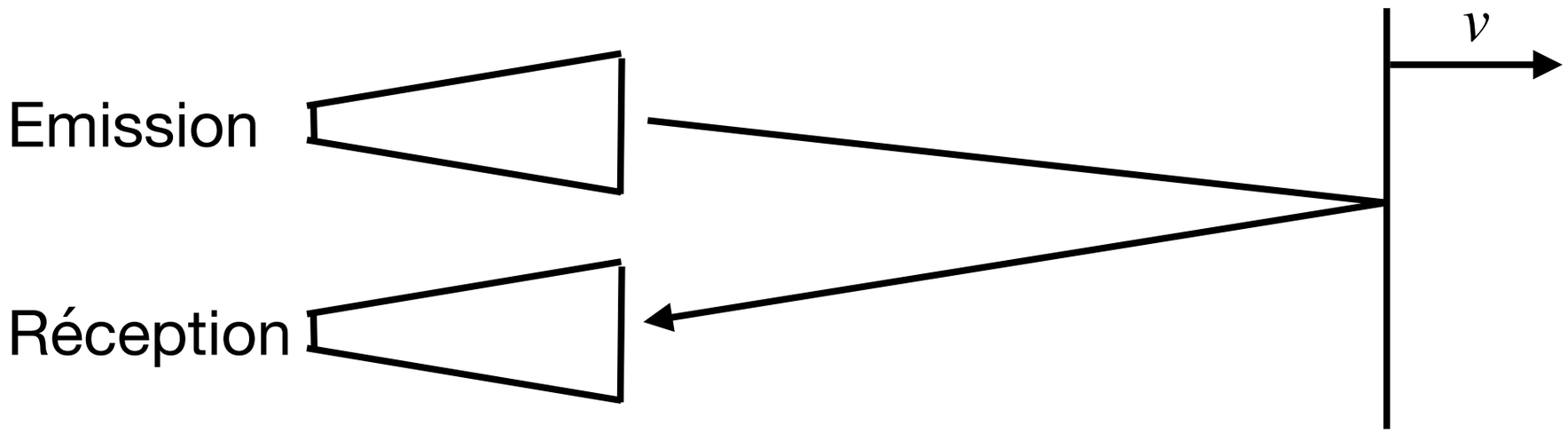
- Ecrivez en notation complexe le déphasage entre émission et réception pour une onde monochromatique

- Le radar effectue N mesures à des fréquences $\omega_{\mathrm{LO}} + n\delta\omega$. Ecrire le vecteur des N déphasages sous la forme $[e^{i\phi_1}, \ldots, e^{i\phi_N}]$

- Que donne la FFT de ce vecteur ?

- Ecrivez un programme permettant de visualiser les echos en fonction de leur distance

Radar Doppler



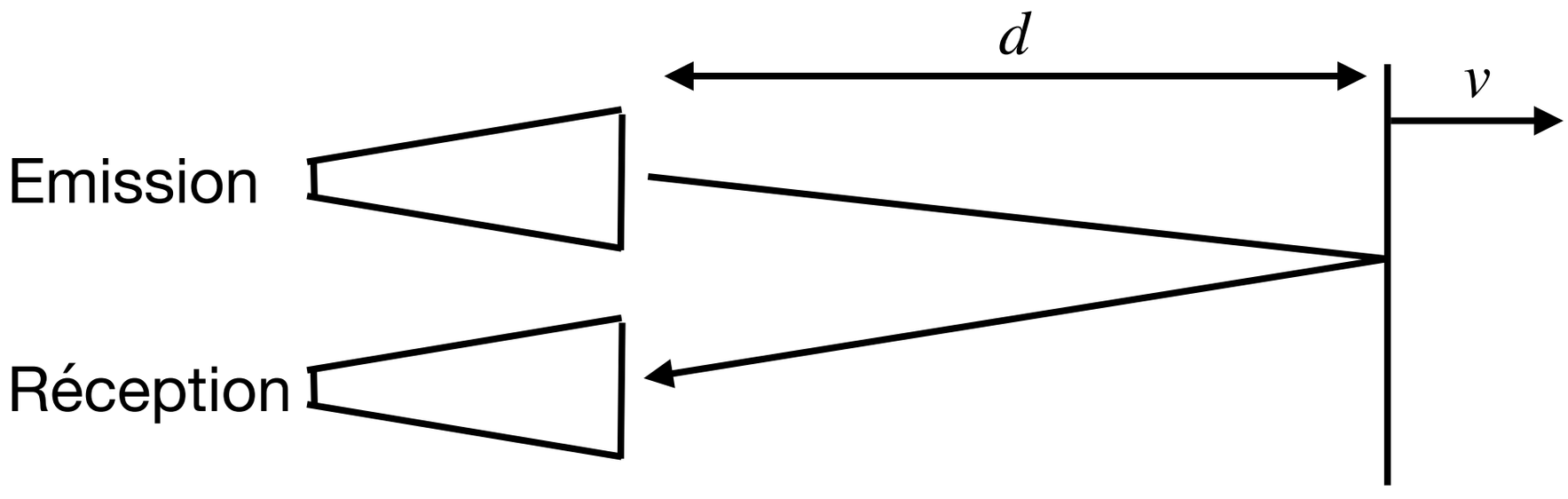
- Introduire le décalage Doppler ω_D

- Le radar effectue maintenant N mesures à une unique fréquence $\omega_{\mathrm{LO}} + \omega_{\mathrm{IF}}$ à des temps $n\delta t$. Ecrire le vecteur des N mesures après demodulation sous la forme $[e^{i\phi_1}, \ldots, e^{i\phi_N}]$

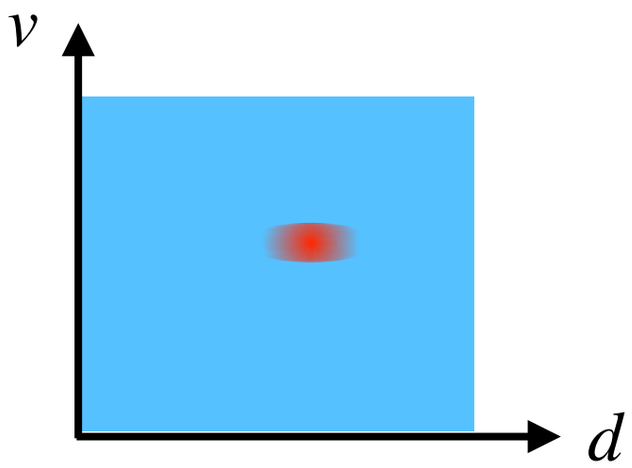
- Que donne la FFT de ce vecteur ?

- Ecrivez un programme permettant de visualiser les echos en fonction de leur vitesse

Radar Position / Vitesse



- Combinez les deux approches précédentes pour afficher sur une image 2D les echos en fonction de leur position et vitesse



https://wirelesspi.com/fmcw-radar-part-1-ranging/

https://wirelesspi.com/fmcw-radar-part-2-velocity-angle-and-radar-data-cube/

https://wirelesspi.com/fmcw-radar-part-3-design-guidelines/