Using Qorvo DWM3001CDK for Indoor Tracking

1 Overview

This document provides instructions to set up Qorvo DWM3001CDK modules to track devices indoors as a GPS replacement, particularly useful in underground or GNSS-denied areas. One module will act as the **initiator** and collect distance data from multiple **responders**.

2 Preparation for All Boards

2.1 Clone Firmware

Ensure you have Git installed. If not:

- On Ubuntu/Debian: `sudo apt install git`
- On Arch: `sudo pacman -S git`
- On Windows: Install Git from `https://git-scm.com/`

Clone the demo firmware:

```
git clone https://github.com/Uberi/DWM3001CDK-demo-firmware
cd DWM3001CDK-demo-firmware
```

2.2 Use Docker Environment (Optional)

This avoids building the toolchain manually:

```
docker pull uberi/qorvo-nrf52833-board
```

2.3 Build Firmware

```
make build
```

2.4 Flash Firmware

Connect the **lower USB port** (J9) to your PC:

```
make flash
```

2.5 Open Serial Terminal

Disconnect J9 and connect the **upper USB port** (J20). Then run:

```
make serial-terminal
```

Or manually via:

- Linux: `screen /dev/ttyACMX 115200` (find device via `ls /dev/ttyACM*`)
- Windows: use PuTTY and connect to COM port at 115200 baud

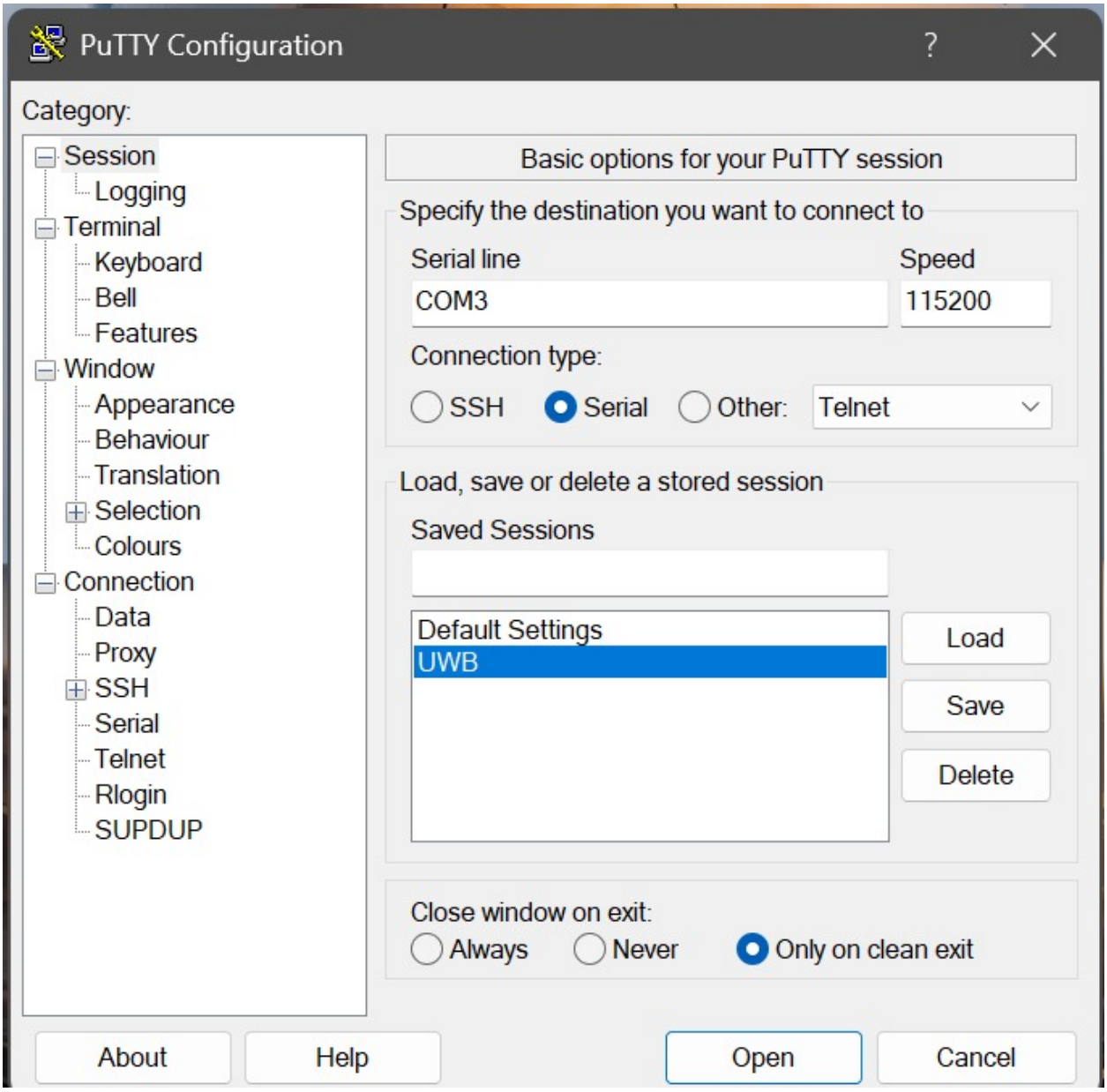

Figure 1: Example PuTTY settings

3 Configure the Initiator and Responders

3.1 Initiator Command (Example)

```
initf 4 2400 200 25 2 42 01:02:03:04:05:06:07:08 1 0 0 1 2
SAVE
```

3.2 Responder Command (Example)

```
respf 4 2400 200 25 2 42 01:02:03:04:05:06:07:08 1 0 0 2
SAVE
```

3.3 Explanation of Command Format

`initf <channel> <period_ms> <slot_duration_ms> <slots> <pac> <network_id> <pan_id> <multinode> 0 0 <initiator_id> <responder_id>`
`respf <channel> <period_ms> <slot_duration_ms> <slots> <pac> <network_id> <pan_id> <multinode> 0 0 <responder_id>`

Parameter-by-Parameter Breakdown:

- `channel` — UWB channel (usually 4 or 5).
- `period_ms` — How often (in ms) the initiator restarts the ranging cycle (e.g., 2400).
- `slot_duration_ms` — Duration of each time slot per responder (e.g., 200).
- `slots` — Total number of reserved time slots (e.g., 25).
- `pac` — Preamble acquisition chunk size (e.g., 2).
- `network_id` — Shared 8-bit network ID (e.g., 42).
- `pan_id` — Shared 64-bit PAN ID (e.g., 01:02:03:04:05:06:07:08).
- `multinode` — Set to 1 to enable multi-responder mode.
- `0 0` — Reserved parameters (must be set to 0).
- `initiator_id` — Address of the initiator.
- `responder_id` — Address of the responder.

You should see `ok` after each command.

4 Running the Setup

- Power responders via battery or USB adapter.
- Keep the initiator connected to your computer.

To view the data:

- Linux: `screen /dev/ttyACMX 115200`
- Windows: open COM port in PuTTY (115200 baud)

You should see blocks of JSON data like:

```
{"Block":286, "results":[{"Addr":"0x0001", "Status":"Ok", "D_cm":12, ...}]}
```

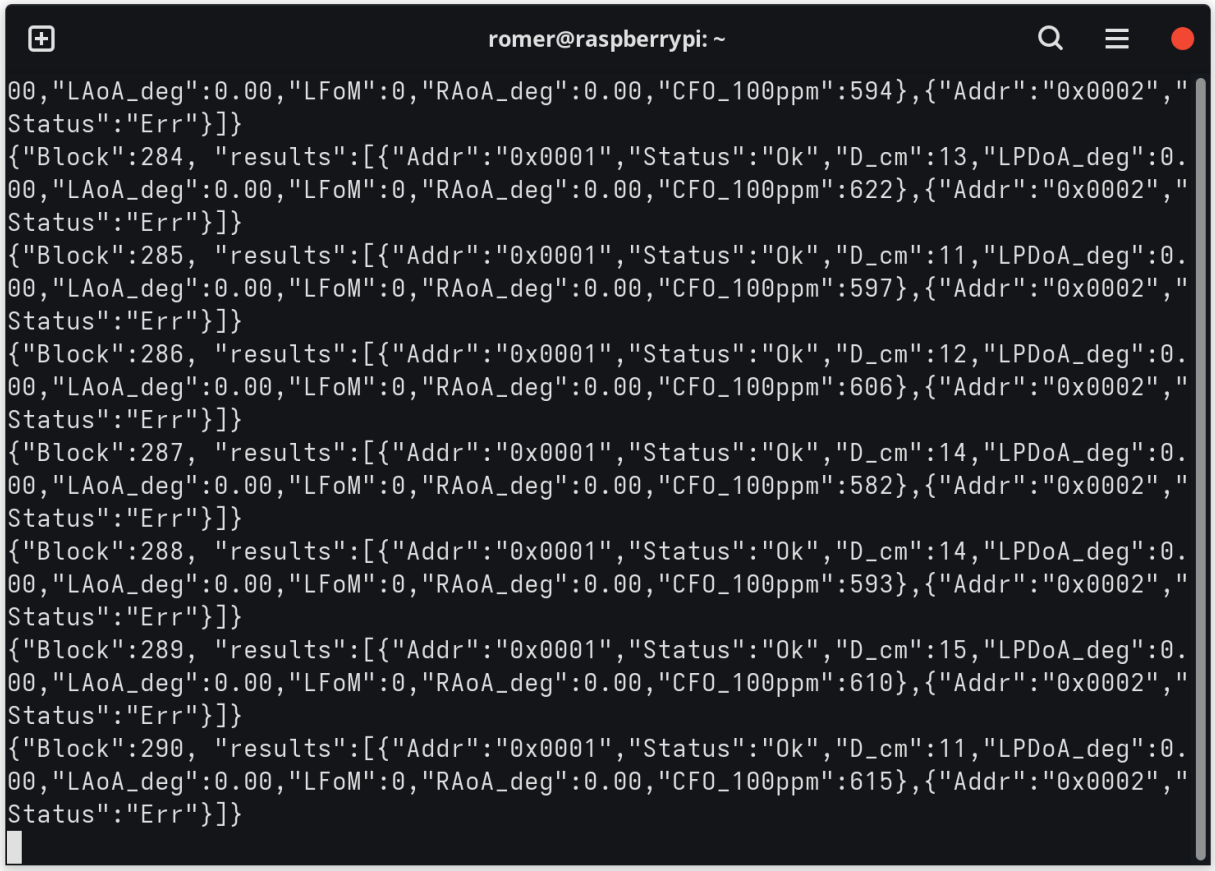

Figure 2: Example output

5 Accessing Data in Python (Basic Script)

```
import serial
import json

ser = serial.Serial('/dev/ttyACM0', 115200, timeout=1)

while True:
    try:
        line = ser.readline().decode('utf-8').strip()
        if line.startswith('{') and line.endswith('}'):
            data = json.loads(line)
            print(data)
    except KeyboardInterrupt:
        break
ser.close()
```

6 Advanced Python Script with Filtering

Install requirements:

```
pip install pyserial numpy numba
```

```
import serial, json, sys
import numpy as np
from collections import deque
from numba import njit
import serial.tools.list_ports

def find_serial_port():
    for port, _, _ in serial.tools.list_ports.comports():
        if "ACM" in port or "USB" in port:
            return port
    return None

PORT = find_serial_port()
if not PORT:
    sys.exit("No␣suitable␣port␣found")

ser = serial.Serial(PORT, 115200, timeout=1)
distance_history = {}
spike_streak = {}
WINDOW, MAX_JUMP, RESET_AFTER = 10, 25, 3

@njit
def moving_average(arr): return np.mean(arr)
@njit
def check_jump(last, current, threshold): return abs(current - last) > threshold

try:
    while True:
        try:
            line = ser.readline().decode().strip()
            if not line.startswith('{'): continue
            data = json.loads(line)
            for result in data.get("results", []):
                addr, status, dist = result.get("Addr"), result.get("Status"), result.get("D_cm")
                if status != "Ok": continue

                if addr not in distance_history:
                    distance_history[addr] = deque([dist], maxlen=WINDOW)
                    spike_streak[addr] = 0
                else:
                    last = distance_history[addr][-1]
                    if check_jump(last, dist, MAX_JUMP):
                        spike_streak[addr] += 1
                        if spike_streak[addr] >= RESET_AFTER:
                            half = WINDOW // 2
                            recent = list(distance_history[addr])[:half]
                            distance_history[addr] = deque([dist]*half + recent, maxlen=WINDOW)
                            spike_streak[addr] = 0
                        else:
                            dist = last * 0.7 + dist * 0.3
                            distance_history[addr].append(dist)
                    else:
                        spike_streak[addr] = 0
                        distance_history[addr].append(dist)

                avg = moving_average(np.array(distance_history[addr]))
                sys.stdout.write(f"\r[{addr}]␣{avg:.1f}␣cm␣␣␣␣")
                sys.stdout.flush()
        except (json.JSONDecodeError, UnicodeDecodeError):
            continue
except KeyboardInterrupt:
    print("\nStopped␣by␣user.")
```

7 Advanced Python Script with Kalman Filter

This script uses a Kalman filter, a powerful algorithm ideal for tracking moving objects. Instead of just averaging recent measurements, it builds a model of the object's motion (its distance and velocity). This allows it to make intelligent predictions, resulting in much smoother, more accurate tracking that is less affected by random measurement errors.

Install requirements:

```
pip install pyserial numpy numba filterpy
```

```
import serial
import json
import sys
import time
from collections import deque
import numpy as np
import serial.tools.list_ports
from filterpy.kalman import KalmanFilter

# -- Configuration --
# UWB Settings
BAUD_RATE = 115200

# Kalman Filter Settings
# Time step (how often we get new data). Assume ~10Hz, but we'll calculate it dynamically.
INITIAL_DT = 0.1
# Measurement uncertainty (how much we trust the UWB reading). Higher means less trust.
MEASUREMENT_NOISE = 10
# Process uncertainty (how much we trust our prediction model). Higher means the model
# expects the object's velocity to change more erratically.
PROCESS_NOISE = 0.1

# -- Helper Functions --
def find_serial_port():
    """Dynamically find the correct serial port."""
    ports = serial.tools.list_ports.comports()
    for port, desc, hwid in sorted(ports):
        if "ACM" in port or "USB" in port or "VCP" in port:
            print(f"Found␣suitable␣port:␣{port}")
            return port
    return None

def create_kalman_filter():
    """Create a 1D Kalman filter for tracking distance and velocity."""
    kf = KalmanFilter(dim_x=2, dim_z=1)
    # State vector: [distance, velocity]
    kf.x = np.zeros((2, 1))
    # State Transition Matrix: predicts the next state based on current state
    # [1, dt]
    # [0, 1]
    kf.F = np.array([[1., INITIAL_DT],
                     [0., 1.]])
    # Measurement Function: maps the state to the measurement
    # [1, 0]
    kf.H = np.array([[1., 0.]])
    # Measurement Noise Covariance
    kf.R = np.array([[MEASUREMENT_NOISE]])
    # Process Noise Covariance
    # We use Q_discrete_white_noise to generate the covariance matrix
    # from our single process noise value and time step.
    from filterpy.common import Q_discrete_white_noise
    kf.Q = Q_discrete_white_noise(dim=2, dt=INITIAL_DT, var=PROCESS_NOISE)
    return kf
```

```
# -- Main Application --
if __name__ == "__main__":
    PORT = find_serial_port()
    if PORT is None:
        print("Error: No suitable serial port found. Exiting.")
        sys.exit(1)

    # Dictionary to hold a separate Kalman filter for each responder address
    kalman_filters = {}
    last_time = {}

    try:
        with serial.Serial(PORT, BAUD_RATE, timeout=1) as ser:
            print(f"Successfully connected to {PORT} at {BAUD_RATE} baud.")
            print("Starting UWB data acquisition... Press Ctrl+C to stop.")

            while True:
                try:
                    line = ser.readline().decode("utf-8").strip()
                    if not line.startswith("{"):
                        continue

                    data = json.loads(line)
                    results = data.get("results", [])
                    display_lines = []

                    for result in results:
                        addr = result.get("Addr")
                        status = result.get("Status")
                        dist = result.get("D_cm")

                        if not addr or status != "Ok" or dist is None:
                            continue

                        current_time = time.time()

                        # Initialize a new filter if we see a new address
                        if addr not in kalman_filters:
                            print(f"\nNew responder detected: {addr}. Initializing filter.")
                            kalman_filters[addr] = create_kalman_filter()
                            # Initialize the filter's state with the first measurement
                            kalman_filters[addr].x[0] = dist
                            last_time[addr] = current_time
                            continue

                        # Calculate dynamic time step (dt)
                        dt = current_time - last_time[addr]
                        if dt == 0: continue # Avoid division by zero
                        last_time[addr] = current_time

                        # Update the filter's state transition matrix and noise with the new dt
                        kf = kalman_filters[addr]
                        kf.F[0, 1] = dt
                        from filterpy.common import Q_discrete_white_noise
                        kf.Q = Q_discrete_white_noise(dim=2, dt=dt, var=PROCESS_NOISE)

                        # --- Kalman Filter Steps ---
                        # 1. Predict the next state
                        kf.predict()
                        # 2. Update the state with the new measurement
                        kf.update(np.array([[dist]]))

                        filtered_dist = kf.x[0, 0]
                        velocity = kf.x[1, 0] # in cm/s

                        display_lines.append(f"[{addr}] Dist: {filtered_dist:.1f} cm (V: {velocity:.1f} cm/s)")

                    if display_lines:
```

```
                    sys.stdout.write("\r" + "␣|␣".join(display_lines) + "␣" * 10)
                    sys.stdout.flush()

            except (json.JSONDecodeError, UnicodeDecodeError):
                # Silently ignore lines that are not valid JSON
                continue
            except Exception as e:
                print(f"\nAn␣unexpected␣error␣occurred:␣{e}")


    except serial.SerialException as e:
        print(f"\nSerial␣Error:␣{e}.␣Please␣check␣the␣connection.")
    except KeyboardInterrupt:
        print("\n\nProgram␣stopped␣by␣user.")
    finally:
        sys.exit(0)
```