Super Slim and Low Power Radar-based Gesture Recognition

Stephan Schoenfeldt, Maximilian Strobel, Jianyu Zhao, Jonas Daugalas

Infineon Technologies AG, Am Campeon 1-15, 85579 Neubiberg, Germany

Summary

This poster presents implementation for embedded radar-based gesture detection, featuring enhanced performance and reduced computational requirements, silicon usage, and power consumption compared to existing methods. We assess both established and emerging tools and techniques for embedded pre-processing, as well as neural network mapping and edge inference.

Previous work

In the past, we introduced an approach that transformed radar data into visual representations and utilized a convolutional neural network (CNN) inspired by computer vision techniques to classify various hand gestures. Our solution achieved a commendable accuracy of 90% for three distinct hand gestures within a maximum range of 30 cm.

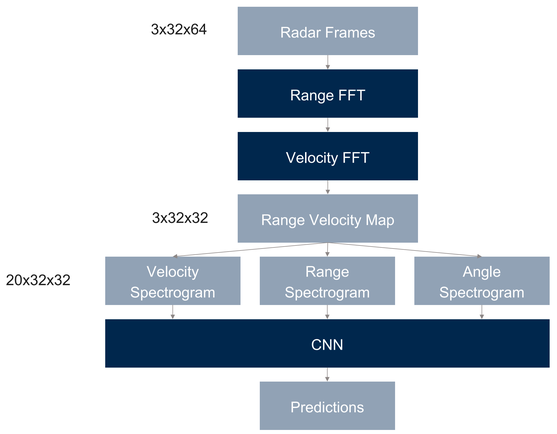

Figure 1. Vision inspired classification algorithm

Challenges and Shortcomings in previous Work

Due to limitations in CPU bandwidth, the previous algorithm couldn't support running the CNN-based gesture classifier for every frame on an embedded processor. Instead, we devised specific logic to detect motion in front of the radar sensor, and gesture classification was only performed when motion was detected.

However, this approach has a significant drawback. The motion detection mechanism needs to be fine-tuned for different positions, hand sizes, velocities, and types of gestures. It becomes challenging to find an optimal setting that satisfies application requirements such as sensitivity, robustness, and latency.

Upon triggering gesture detection, a substantial amount of data (the entire history) needs to be processed by a CNN, resulting in noticeable latency for the detection process.

Ultimately, we were able to execute the algorithm on an Arm Cortex-M4 processor running at 150 MHz clock speed, with a CPU utilization of 70%. The RAM usage amounted to 260 kB.

Novel Algorithm

The novel algorithm detects the nearest moving target in front of the sensor by analysing its trajectory over time in five dimensions: range, velocity, azimuth, elevation, and amplitude. This approach efficiently captures the most crucial information for gesture recognition. Consequently, it minimizes the input size for the neural network after pre-processing, enabling more efficient network designs. By utilizing a recurrent neural network (RNN), each frame received from the sensor can be processed promptly. The RNN's hidden state vector retains relevant historical data, eliminating the need for specific logic to detect gesture start and end frames. For every frame, the RNN produces a probability distribution for each supported gesture plus background. A final filtering step transforms this continuous stream of probabilities into discrete gesture events.

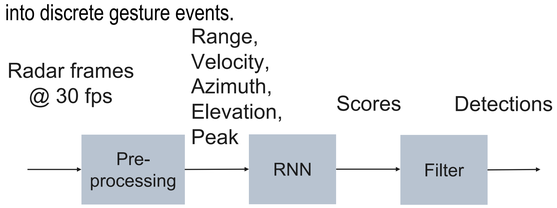

Figure 2. Novel Algorithm Overview

The image below depicts the 5 gestures our algorithm supports. They are detected at a rage up to 1m and with a field of view of +-45° relative to the sensor.

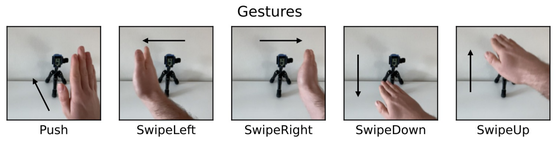

Figure 3. Supported Gestures

Radar Preprocessing

The image displayed below illustrates the primary pre-processing scheme. It predominantly relies on generic digital signal processing (DSP) blocks, which are commonly found in embedded libraries such as ARM's CMSIS-DSP. This enables straightforward and efficient implementation on Cortex M Microcontrollers.

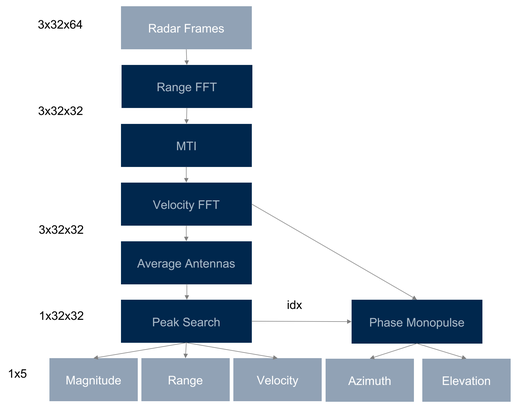

Figure 4. Block Diagram of the Radar Pre-processing

Features

The provided chart depicts the features generated by the pre-processing algorithm for various supported gestures. The average is represented by the blue line, while the pale blue area represents the distribution within our training database.

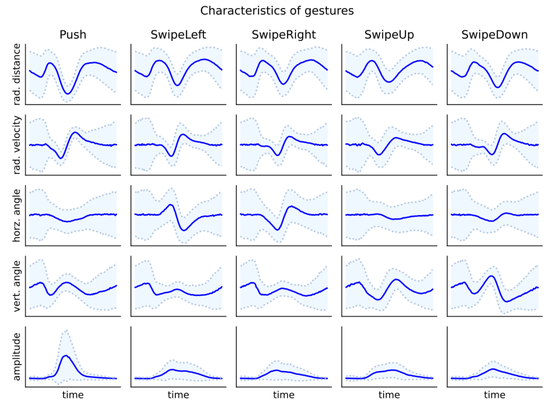

Figure 5. Time Series Features for different Gestures

Neural Network Architecture

The diagram below illustrates the neural network architecture. Thanks to our thorough pre-processing and the compact size of the input features, we can employ a relatively small neural

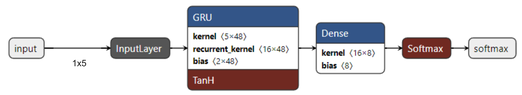

Figure 6. Neural Network Block Diagram

NN Inference Runtimes

We evaluate different inference runtimes with respect to their memory footprint and compute bandwidth for different precisions:

- TensorFlow Lite Micro
- Apache Micro TVM
- TensorFlow Lite Micro Interpreter less (Infineon Fork)
- Octopus (Infineon Inhouse)

Float32 Precision				
	Tensorflow Lite Micro		Tensorflow Lite Micro Interpreterless	
	PSoC6 Cortex-M4F	PSoC4 Cortex-M0+	PSoC6 Cortex-M4F	PSoC4 Cortex-M0+
Flash [kB]	83,40	91,50	32,20	58,70
RAM [kB]	7,00	7,00	4,00	7,00
time [ms]	0,33	13,90	0,29	13,90
	Apache TVM		Octopus	
Flash [kB]	8,40	24,00	9,00	13,00
RAM [kB]	0,75	0,54	0,54	0,50
time [ms]	0,28	16,90	1,47	20,10

Int16 Precision				
	Apache TVM		Octopus	
	PSoC6 Cortex-M4F	PSoC4 Cortex-M0+	PSoC6 Cortex-M4F	PSoC4 Cortex-M0+
Flash [kB]	7,70	10,2	6,50	7,50
RAM [kB]	0,63	0,64	0,43	0,69
time [ms]	0,35	10,40	1,45	12,80

Table 1. Resource usage of different Inference Engines

Target Platforms

The gesture detection application is implemented on two embedded microcontroller systems, each offering different performance and price points.

- PSOC6
  - Cortex-M4F @ 150Mhz
  - 1MB Flash
  - 256kB RAM
- PSOC4
  - Cortex-M0p @ 48Mhz
  - 128 kB Flash
  - 16k B RAM

Figure 7. Application Target Platforms

The provided table presents the memory footprints and frame execution times for varying neural network and pre-processing precisions using different inference runtimes. The numbers represent overall resource usage including pre-processing, neural network inference and application code.

	Preprocessing	Neural Network		FLASH [kB]]	RAM [kB]	CPU [ms]
Processor	Precision	Engine	Precision			
CM4	float32	TFLM	float32	278	25	1,3
CM4	float32	TVM	float32	204	21	1,2
CM4	int16	TVM	float32	121	18	1,4
CM4	int16	TVM	int16	120	18	1,4
CM0	int16	TVM	int16	107	9,7	22

Table 2. Application Level Resource usage

Power consumption

The average power consumption on the Psoc6 Platform remains below 10mW, as illustrated in the chart below, which displays the individual contributors to power consumption.

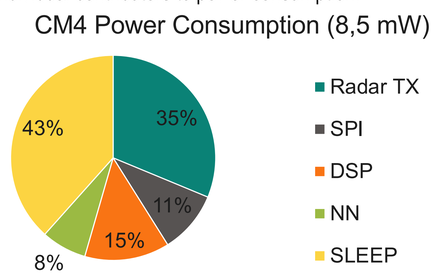

Figure 8. Application Power Consumption

Conclusions

- In this work we demonstrate an embedded radar-based hand gesture recognition system capable of classifying 5 distinct gestures with 96% accuracy on our evaluation dataset and runnable on an ARM Cortex-M0+ based microcontroller.
- The system requires <10 kB RAM, <128 kB FLASH, and consumes <10mW of power.
- Microcontroller with an ARM Cortex-M0+ core clock at 48 MHz can run our algorithm at 30 fps while consuming 73% of the CPU time.

This system is the first to our knowledge to achieve such low memory, storage, compute and power requirements while maintaining high gesture classification accuracy.