دانشگاه صنعتی امیرکبیر
دانشکده کامپیوتر

فاز نهایی پروژه

درس سیستم‌های قابل بازپیکربندی

استاد:
دکتر مرتضی صاحب الزمانی

دانشجویان:
سید حسین ملکوتی
فاطمه فرجلو

شماره دانشجویی:
۴۰۰۱۳۱۹۲۰
۹۹۱۳۱۰۲۰

تیر ۱۴۰۱

نام پروژه :

سیستم بینایی ماشین و کنترلر یک ربات با استفاده از FPGA خانواده ZYNQ

مقدمه :

در این پروژه که یک سیستم بینایی ماشین ساده با استفاده از FPGA پیاده‌سازی شده و از نتایج حاصل از آن برای کنترل یک ربات استفاده می‌شود. در این پروژه از الگوریتم‌های مختلف پردازش تصویر جهت این امر استفاده شده است. گام‌های انجام این پردازش به صورت زیر می‌باشد.

۱- انجام تنظیمات جهت پیکربندی سیستم برای دریافت تصاویر

a. انتخاب ورودی از دوربین با اتصال USB و یا HDMI

۲- بافر نمودن یک فریم

۳- انجام پردازش

۴- تصمیم‌گیری در خصوص عملکرد ربات

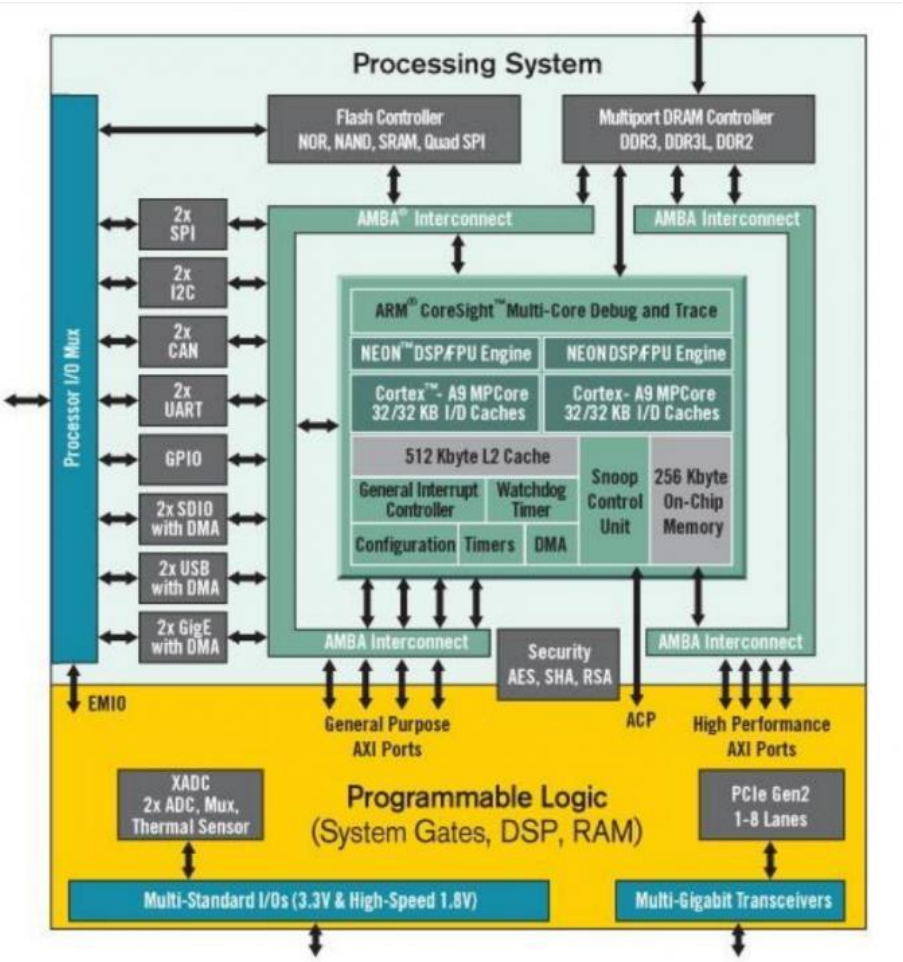
۵- فعال‌سازی موتورها در صورت نیاز

۶- انجام مجدد حلقه

همانگونه که ذکر شد برای انجام این امر از PYNQ[1] استفاده شده است. با استفاده از PYNQ می‌توان برای تراشه‌های قابل برنامه‌ریزی سری Zynq و Zynq Ultrascale شرکت AMD از زبان برنامه‌نویسی Python برای ارتباط با FPGA استفاده نمود.

در خانواده تراشه‌های Zynq دو بخش کلی پردازشی وجود دارند. این دو بخش به اختصار PL[2] و PS[3] نامیده می‌شوند. این معماری در شکل زیر نمایش داده شده است :

[1] - Python productivity for Zynq
[2] - Programmable Logic
[3] - Processing Subsystem



این معماری انعطاف‌پذیری بالایی برای پیاده‌سازی سیستم‌های مختلف پردازشی را ایجاد می‌نماید اما برای انجام این پیاده‌سازی نیاز به دانش فنی عمیق در خصوص معماری داخلی این تراشه‌ها و همچنین توانایی پیاده‌سازی سخت‌افزاری الگوریتم جهت افزایش سرعت اجرای آن می‌باشد. این مساله با ایجاد مواردی همچون قابلیت HLS[4] تا حدی بهبود یافته است اما بازهم امکان استفاده مناسب از این تراشه‌ها، نیازمند موارد یاد شده قبلی می‌باشد. هدف از پروژه PYNQ ایجاد یک بستر مناسب جهت استفاده از این تراشه‌ها جهت پیاده‌سازی الگوریتم‌های مختلف با استفاده از زبان پرکاربرد Python می‌باشد. این قابلیت باعث می‌شود بدون نیاز به نگرانی در خصوص نحوه ارتباطی بین بخش‌های مختلف بتوان یک الگوریتم را پیاده‌سازی نمود و سپس آنرا بین سخت‌افزار PLو PS تقسیم نمود. این قابلیت باعث می‌گردد در اکثر موارد مقایسه بین حالت‌های مختلف پیاده‌سازی سخت‌افزار و نرم‌افزار به سادگی امکان‌پذیر باشد. در حالتی که الگوریتم مورد نظر نیازمند تغییرات فراوان جهت اجرای بهینه باشد این ویژگی بسیار پرکاربرد خواهد

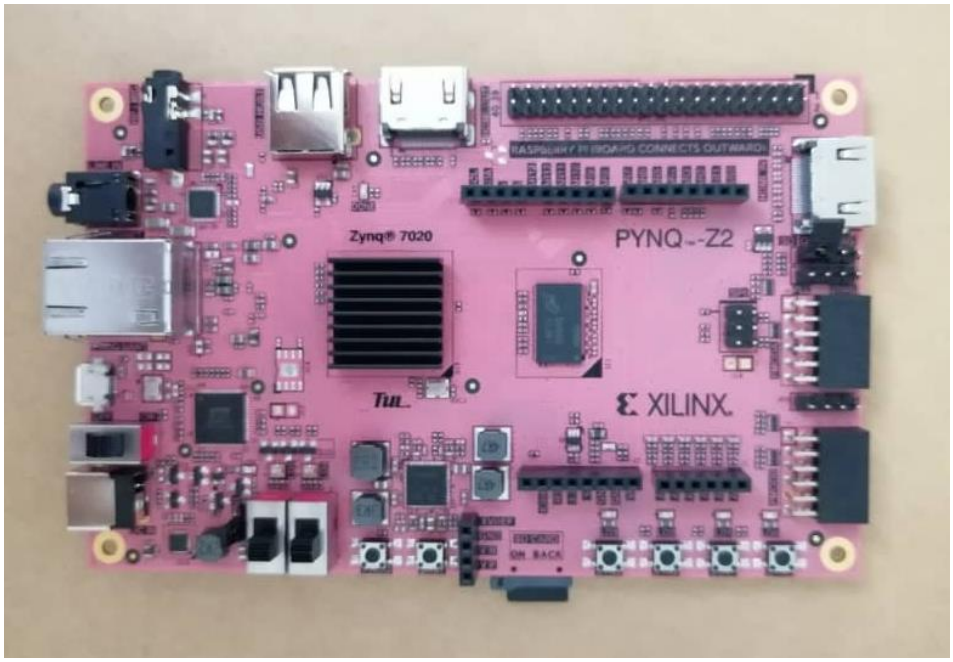
[4] - High Level Synthesis

بود. سرعت بخشی سخت‌افزاری با استفاده از مفهومی به نام Overlay ها صورت می‌گیرد که معمولا در ابتدای برنامه به صورت یک فایل Bit ایجاد شده در ابزارهای مختلف سنتز Load شده و قسمت PL با استفاده از آن برنامه‌ریزی می‌گردد. سپس می‌توان با فراخوانی آن بخش و ارسال مقادیر به آن باعث افزایش سرعت انجام بخش مورد نظر الگوریتم گردید. نکته مهم در این پروسه انتخاب مسیرهای مناسب جهت دریافت داده و ارسال آن می‌باشد. به عنوان مثال در یک پردازش تست در صورت بارگذاری تصاویر از دوربین USB و ارسال آن به قسمت PL به دلیل نیاز به انجام دریافت داده دوربین از قسمت PS و سپس ارسال آن به قسمت PL سرعت پردازشی حدودا ۲.۵ برابر نسبت به حالت دریافت مستقیم داده تصاویر از ورودی HDMI توسط بخش PL کاهش می‌یابد. این مساله در تمامی مسائل پردازشی صادق است و بایستی مسیر دریافت داده تا حد ممکن به بخش پردازشی نزدیک باشد. مساله مهم در PYNQ مدیریت ارتباط بین سخت افزار و نرم‌افزار به صورت خودکار است که توسط توابع مختلف Python انجام می‌گردد.

در این پروژه جهت افزایش سرعت از یک برد Raspberry Pi جهت اخذ تصاویر و تبدیل به HDMI استفاده شده است. در حقیقت نقش این برد تنها یک دوربین HDMI است و پردازشی بر روی داده‌ها انجام نخواهد داد.

از یک مبدل ارزان قیمت AV-HDMI نیز برای این منظور استفاده گردید اما خروجی تصویر کیفیت چندان مناسبی جهت استفاده نداشته است. تصویر این مبدل در زیر آورده شده است :

واحد پردازشی اصلی در این سیستم برد PYNQ-Z2 می‌باشد. تصویر این برد در شکل زیر آورده شده است :



جهت استفاده از این سیستم به یک کارت حافظه نیاز می‌باشد که علاوه بر PYNQ ، Overlay های مورد استفاده نیز بر روی آن قرار گرفته باشد. از آنجا که در این پروژه از Overlay های پردازش تصویر شرکت AMD استفاده شده است این موارد بر روی کارت حافظه قرار داده شده‌اند. نحوه انجام این عمل در بخش بعدی شرح داده شده است.

مراحل اجرایی :

در این مرحله یک کارت حافظه با سرعت بالا و ظرفیت 16GB برای اتصال به برد استفاده شده و ایمیج مربوطه بر روی آن با استفاده از Win32DiskImager نوشته می‌شود. سپس با اتصال یک پاور مناسب (بین ۷ تا ۱۵ ولت پیشنهاد شده ۱۲ ولت) برد روشن شده و پس از بوت لینوکس آماده استفاده می‌گردد. بایستی توجه نمود که آخرین نسخه ایمیج PYNQ نسخه ۲.۷ می‌باشد (در زمان نگارش این متن) که با بسیاری از Overlay ها و درایورهای آن‌ها سازگاری لازم را ندارد لذا برای حفظ سازگاری از نسخه ۲.۶ استفاده شده است.

در این مرحله بایستی Overlay های پردازش تصویر بر روی برد نصب گردد. برای این منظور از دستورات زیر استفاده می‌شود (دستورات و خروجی‌ها آورده شده‌اند) :

ابتدا برخی الزامات اولیه و به‌روزرسانی‌ها نصب می‌گردند:

```
xilinx@pynq:~$ sudo apt-get update
[sudo] password for xilinx:
Get:1 http://ports.ubuntu.com/ubuntu-ports bionic InRelease [242 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports bionic/main Sources [829 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports bionic/universe Sources [9,051 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports bionic/main armhf Packages [968 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports bionic/main Translation-en [516 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports bionic/universe armhf Packages [8,269 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports bionic/universe Translation-en [4,941 kB]
Fetched 24.8 MB in 39s (641 kB/s)
Reading package lists... Done
```

```
xilinx@pynq:~$ sudo pip3 install --upgrade cython
Collecting cython
  Downloading Cython-0.29.30-py2.py3-none-any.whl (985 kB)
     |████████████████████████████████| 985 kB 481 kB/s
Installing collected packages: cython
  Attempting uninstall: cython
    Found existing installation: Cython 0.29
    Uninstalling Cython-0.29:
      Successfully uninstalled Cython-0.29
Successfully installed cython-0.29.30

xilinx@pynq:~$ sudo /usr/bin/python3.6 -m pip install --upgrade pip
Collecting pip
Downloading pip-21.3.1-py3-none-any.whl (1.7 MB)
MB 187 kB/s ١.٧ |████████████████████████████████|
Installing collected packages: pip
Attempting uninstall: pip
Found existing installation: pip 20.2.4
:Uninstalling pip-20.2.4
Successfully uninstalled pip-20.2.4
Successfully installed pip-21.3.1
```

با دستور زیر کتابخانه‌های پردازش تصویر بخش PS نصب می‌گردد. به دلیل طولانی بودن خروجی دستور بعدی تنها ابتدا و انتهای آن آورده شده است :

```
sudo apt-get install libopencv-*
…
```

…

Setting up libcv-bridge-dev (1.12.3+ds-2) ...

Setting up libopencv-apps-dev (1.11.14-1build4) ...

Processing triggers for libc-bin (2.27-3ubuntu1) ...

…

…

/sbin/ldconfig.real: /usr/local/lib/libopencv_saliency.so.3.4 is not a symbolic link

با دستور زیر Overlay ها و کتابخانه‌های مربوط به بخش PL پردازش تصویر نصب می‌گردند :

xilinx@pynq:~$ sudo pip3 install git+https://github.com/Xilinx/PYNQ-ComputerVision.git

Collecting git+https://github.com/Xilinx/PYNQ-ComputerVision.git

Cloning https://github.com/Xilinx/PYNQ-ComputerVision.git to /tmp/pip-req-build-v7l0s2vg

Running command git clone --filter=blob:none -q https://github.com/Xilinx/PYNQ-ComputerVision.git /tmp/pip-req-build-v7l0s2vg

Resolved https://github.com/Xilinx/PYNQ-ComputerVision.git to commit f270218cd1d296aa7ecbc927030d0abfd2ad51e3

Running command git submodule update --init --recursive -q

Preparing metadata (setup.py) ... done

Requirement already satisfied: pynq>=2.3 in /usr/local/lib/python3.6/dist-packages (from pynq-cv==2.3)(۲.۶.۰)

Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages (from pynq>=2.3->pynq-cv==2.3)(۰.۲۲.۰)

Requirement already satisfied: cffi in /usr/lib/python3/dist-packages (from pynq>=2.3->pynq-cv==2.3)(۱.۱۱.۵)

Requirement already satisfied: setuptools>=24.2.0 in /usr/lib/python3/dist-packages (from pynq>=2.3->pynq-cv==2.3)(۳۹.۰.۱)

Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from pynq>=2.3->pynq-cv==2.3)(۱.۱۶.۰)

Requirement already satisfied: python-dateutil>=2 in /usr/lib/python3/dist-packages (from pandas->pynq>=2.3->pynq-cv==2.3)(۲.۶.۱)

Requirement already satisfied: pytz>=2011k in /usr/lib/python3/dist-packages (from pandas->pynq>=2.3->pynq-cv==2.3)(۲۰۱۸.۳)

Building wheels for collected packages: pynq-cv

Building wheel for pynq-cv (setup.py) ... done

Created wheel for pynq-cv: filename=pynq_cv-2.3-py3-none-any.whl size=2371523 sha256=7ff1f68610f8a8335576b0b6bff163e43ee9cb48196c3f9a964c676717a3d9eb

Stored in directory: /tmp/pip-ephem-wheel-cache-itj2qcqt/wheels/84/72/b7/db59bfda917760ac888d82e134c11951b9d3cf8e4b3df48afa

Successfully built pynq-cv

Installing collected packages: pynq-cv

Successfully installed pynq-cv-2.3

پس از نصب همه موارد با استفاده از دستور sudo reboot سیستم را مجددا راه‌اندازی می-نماییم. در این مرحله می‌توان با استفاده از Jupyter شروع به نوشتن برنامه نمود.

با توجه به اینکه هر برنامه پردازش تصویر از چندین زیر بخش پردازشی مختلف ایجاد شده است، در ابتدا تست مربوط به هر بخش به صورت خاص انجام می‌پذیرد تا بتوان نتیجه‌گیری مناسبی در خصوص میزان سرعت‌بخشی پیاده‌سازی سخت‌افزار داشت. یکی از پرکاربردترین

عملیات‌ها در پردازش تصویر فیلتر ۳ در ۳ می‌باشد که با توجه به مقادیر این ماتریس عملیات ختلفی بر روی تصویر صورت می‌گیرد. پیاده‌سازی این فیلتر بر روی سخت‌افزار و نرم‌افزار انجام پذیرفته و نتایج با یکدیگر مقایسه می‌گردند.

در تست‌های انجام شده از یک set-top box (STB) به عنوان ورودی تصویر HDMI استفاده شده است.

در ادامه کدهای مربوطه همراه با نتایج حاصله آورده شده است :

تست فیلتر تک:

```
from pynq import Overlay
import PIL.Image
import numpy as np
import time
import cv2

bareHDMI = Overlay("/usr/local/lib/python3.6/dist-packages/pynq_cv/overlay
s/xv2Filter2DDilate.bit")
import pynq_cv.overlays.xv2Filter2DDilate as xv2
from pynq import Xlnk
from pynq.lib.video import *
Xlnk.set_allocator_library("/usr/local/lib/python3.6/dist-packages/pynq_cv
/overlays/xv2Filter2DDilate.so")
mem_manager = Xlnk()
hdmi_in = bareHDMI.video.hdmi_in
hdmi_out = bareHDMI.video.hdmi_out

hdmi_in.configure(PIXEL_GRAY)
hdmi_out.configure(hdmi_in.mode)

hdmi_in.cacheable_frames = False
hdmi_out.cacheable_frames = False

hdmi_in.start()
hdmi_out.start()
```

```
/usr/lib/python3/dist-packages/ipykernel_launcher.py:12: DeprecationWarnin
g: pynq.Xlnk is deprecated and will be removed in 2.7 - use pynq.allocate
instead
  if sys.path[0] == '':
```

```
<contextlib._GeneratorContextManager at 0xaffd8830>
```

```
mymode = hdmi_in.mode
print("HDMI Pic Spec: "+str(mymode))
```

```
height = hdmi_in.mode.height
width = hdmi_in.mode.width
bpp = hdmi_in.mode.bits_per_pixel
```

```
HDMI Pic Spec: VideoMode: width=1280 height=720 bpp=8
```

```
hdmi_in.tie(hdmi_out)
```

```
#Sobel filter
kernelF = np.array([[1.0,0.0,-1.0],[2.0,0.0,-2.0],[1.0,0.0,-1.0]],np.float
32)
buf     = np.ones((height,width),np.uint8)
numframes = 60

start = time.time()
for _ in range(numframes):
    inframe = hdmi_in.readframe()
    outframe = hdmi_out.newframe()
    cv2.filter2D(inframe, -1, kernelF, dst=outframe)
    inframe.freebuffer()
    hdmi_out.writeframe(outframe)
end = time.time()
print("Frames per second:  " + str(numframes / (end - start)))
```

Frames per second: 7.409681836059334

```
kernelF = np.array([[1.0,0.0,-1.0],[2.0,0.0,-2.0],[1.0,0.0,-1.0]],np.float
32)

numframes = 600

start = time.time()
for _ in range(numframes):
    inframe = hdmi_in.readframe()
    outframe = hdmi_out.newframe()
    xv2.filter2D(inframe, -1, kernelF, dst=outframe, borderType=cv2.BORDER
_CONSTANT)
    inframe.freebuffer()
    hdmi_out.writeframe(outframe)
end = time.time()
print("Frames per second:  " + str(numframes / (end - start)))
```

Frames per second: 50.03799800717497

```
image = PIL.Image.fromarray(inframe)
image
```


```
import PIL.Image
image = PIL.Image.fromarray(outframe)
image
```


```
hdmi_out.close()
hdmi_in.close()
```

تست دو فیلتر:

```
from pynq import Overlay
import PIL.Image
import numpy as np
import time
import cv2

bareHDMI = Overlay("/usr/local/lib/python3.6/dist-packages/pynq_cv/overlay
s/xv2Filter2DDilate.bit")
import pynq_cv.overlays.xv2Filter2DDilate as xv2
from pynq import Xlnk
from pynq.lib.video import *
Xlnk.set_allocator_library("/usr/local/lib/python3.6/dist-packages/pynq_cv
/overlays/xv2Filter2DDilate.so")
mem_manager = Xlnk()
hdmi_in = bareHDMI.video.hdmi_in
hdmi_out = bareHDMI.video.hdmi_out

hdmi_in.configure(PIXEL_GRAY)
hdmi_out.configure(hdmi_in.mode)

hdmi_in.cacheable_frames = False
hdmi_out.cacheable_frames = False

hdmi_in.start()
hdmi_out.start()
```

```
/usr/lib/python3/dist-packages/ipykernel_launcher.py:12: DeprecationWarnin
g: pynq.Xlnk is deprecated and will be removed in 2.7 - use pynq.allocate
instead
  if sys.path[0] == '':
```

```
<contextlib._GeneratorContextManager at 0xb0058f70>
```

```
mymode = hdmi_in.mode
print("HDMI Pic Spec: "+str(mymode))

height = hdmi_in.mode.height
width = hdmi_in.mode.width
bpp = hdmi_in.mode.bits_per_pixel
```

```
HDMI Pic Spec: VideoMode: width=1280 height=720 bpp=8
```

```
hdmi_in.tie(hdmi_out)

import numpy as np
import time
import cv2

kernelF = np.array([[1.0,0.0,-1.0],[2.0,0.0,-2.0],[1.0,0.0,-1.0]],np.float
32)
kernelD = np.ones((3,3),np.uint8)
buf     = np.ones((height,width),np.uint8)
```

```
numframes = 20

start = time.time()
for _ in range(numframes):
    inframe = hdmi_in.readframe()
    outframe = hdmi_out.newframe()
    cv2.filter2D(inframe, -1, kernelF, dst=buf)
    cv2.dilate(buf, kernelD, dst=outframe, iterations=1)
    inframe.freebuffer()
    hdmi_out.writeframe(outframe)
end = time.time()
print("Frames per second:  " + str(numframes / (end - start)))
```

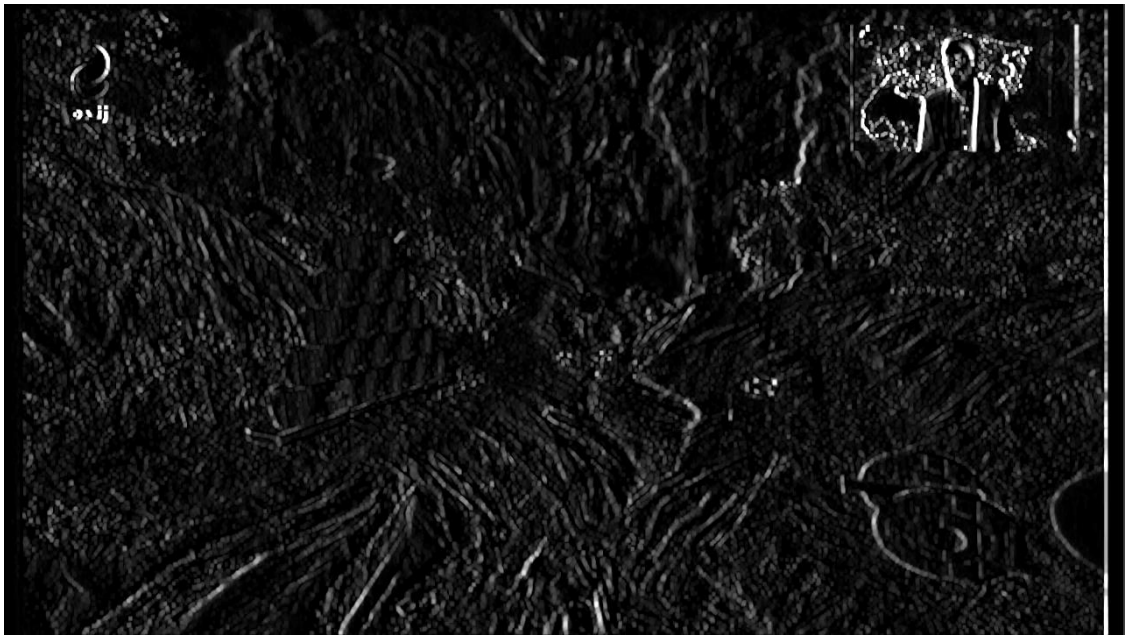
Frames per second: 9.361748667542956

```
kernelF = np.array([[1.0,0.0,-1.0],[2.0,0.0,-2.0],[1.0,0.0,-1.0]],np.float
32)
kernelVoid = np.zeros(0)
xFbuf = mem_manager.cma_array((height,width),np.uint8)
numframes = 600
start=time.time()
for _ in range(numframes):
    inframe = hdmi_in.readframe()
    outframe = hdmi_out.newframe()
    xv2.filter2D(inframe, -1, kernelF, dst=xFbuf,borderType=cv2.BORDER_CON
STANT)
    xv2.dilate(xFbuf, kernelVoid, dst=outframe,borderType=cv2.BORDER_CONST
ANT)
    inframe.freebuffer()
    hdmi_out.writeframe(outframe)
end=time.time()
print("Frames per second:  " + str(numframes / (end - start)))
```

Frames per second: 48.903022486764264

```
image = PIL.Image.fromarray(inframe)
image
```


```
import PIL.Image
image = PIL.Image.fromarray(outframe)
image
```



```
hdmi_out.close()
hdmi_in.close()
```

نکته جالب افزایش سرعت حالت نرم‌افزاری در تست دو فیلتر نسبت به فیلتر تک می‌باشد. پس از بررسی مشخص گردید دلیل این امر اشغال شدن کانال‌های سخت‌افزاری ارتباطی به دلیل ا ستفاده همزمان از ورودی و خروجی HDMI است. این مساله در زیر نمایش داده شده است :

```
inframe = hdmi_in.readframe()
outframe = hdmi_out.newframe()
cv2.filter2D(inframe, -1, kernelF, dst=outframe)
```

در حالت دو فیلتر از یک بافر میانی استفاده شده است که این قضیه باعث بهبود کارایی کد شد ه است. این قطعه کد در زیر نمایش داده شده است :

```
buf      = np.ones((height,width),np.uint8)
…

cv2.filter2D(inframe, -1, kernelF, dst=buf)
cv2.dilate(buf, kernelD, dst=outframe, iterations=1)
```

Test Type	1	2
Hardware	50.03799800717497	48.903022486764264
Software	7.409681836059334	9.361748667542956
Speed Up	6.753056	5.223706

بایستی توجه داشت که این افزایش سرعت‌ها با توجه به سرعت پایه کلاک‌های سیستم ایجاد شده است و امکان تغییر آن با تغییر این کلاک‌ها وجود دارد.

در تست پردازش تصویر برای برنامه ربات از چندین مدل برای پردازش تصویر و هدایت ربات استفاده گردیده است. دو مورد اصلی عبارتند از :

۱- تشخیص حرکت

۲- تشخیص رنگ

در همه موارد سعی گردید که نمونه سخت‌افزاری و نرم‌افزاری، هر دو، مورد تست قرار گیرد. نکته مهم در این خصوص وجود یا عدم وجود فضا و همچنین تابع مورد نظر در Overlay خاص سخت‌افزار مورد استفاده می‌باشد. برای این موضوع بخشی از تابع مورد نظر به صورت نرم‌افزاری و بخشی به صورت سخت‌افزاری مورد بررسی قرار گرفته است. این موارد از نظر پیاده‌سازی بسیار مشابه هستند و تنها ترتیب فیلترها و نحوه استفاده از آن‌ها تا حدی متفاوت است. لذا در این بخش کد تشخیص حرکت و نتایج مربوط به آن بررسی می‌گردد. در این کد بخش‌ی از کد به صورت اجرا بر روی پروسسور و بخش دیگر به صورت

اجرا بر روی بخش Logic انجام پذیرفته است که نتایج آن در ادامه بررسی می‌گردد. شایان ذکر است طی بررسی مدت اجرای هر بخش الگوریتم مشخص گردید که Filter2D تقریبا طولانی ترین زمان اجرا را در بین تمامی موارد دارد، لذا این مورد جهت اجرای سخت‌افزاری با توجه به محدودیت‌های موجود سخت‌افزار انتخاب گردیده است.

```
from pynq import Overlay
import PIL.Image
import numpy as np
import time
import cv2
print("run S")
bareHDMI = Overlay("/usr/local/lib/python3.6/dist-packages/pynq_cv/overlay
s/xv2Filter2DDilate.bit")
import pynq_cv.overlays.xv2Filter2DDilate as xv2
from pynq import Xlnk
from pynq.lib.video import *
Xlnk.set_allocator_library("/usr/local/lib/python3.6/dist-packages/pynq_cv
/overlays/xv2Filter2DDilate.so")
mem_manager = Xlnk()
hdmi_in = bareHDMI.video.hdmi_in
hdmi_out = bareHDMI.video.hdmi_out

hdmi_in.configure(PIXEL_GRAY)
hdmi_out.configure(hdmi_in.mode)

hdmi_in.cacheable_frames = False
hdmi_out.cacheable_frames = False

hdmi_in.start()
hdmi_out.start()
print("run F")
```

```
run S

/usr/lib/python3/dist-packages/ipykernel_launcher.py:12: DeprecationWarnin
g: pynq.Xlnk is deprecated and will be removed in 2.7 - use pynq.allocate
instead
  if sys.path[0] == '':

run F
```

```
mymode = hdmi_in.mode
print("HDMI Pic Spec: "+str(mymode))

height = hdmi_in.mode.height
width = hdmi_in.mode.width
bpp = hdmi_in.mode.bits_per_pixel
```

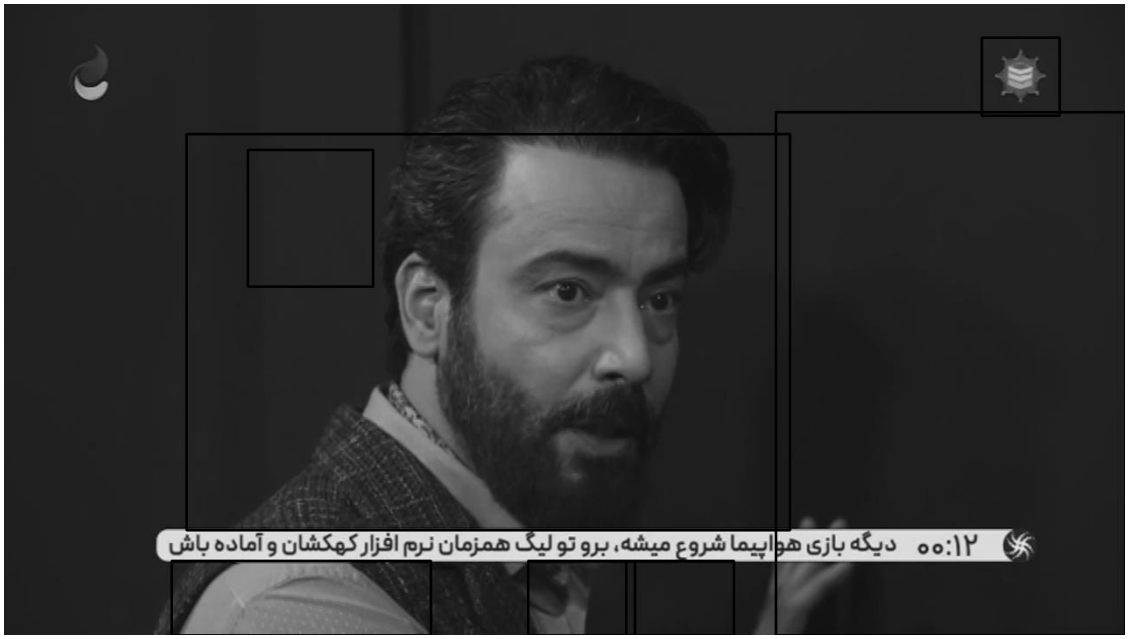
```
HDMI Pic Spec: VideoMode: width=1280 height=720 bpp=8
```

```
hdmi_in.tie(hdmi_out)
```

```
print("Run Fully Software\")

inframe = hdmi_in.readframe()
reference_blur = cv2.GaussianBlur(inframe, (5, 5), 0)
kernelF = np.array([[0.0625,0.125,0.0625],[0.125,0.25,0.125],[0.0625,0.125
,0.0625]], np.float32)
kernelVoid = np.zeros(0)

xFbuf1 = mem_manager.cma_array((height,width),np.uint8)
xFbuf2 = mem_manager.cma_array((height,width),np.uint8)
numframes =10
outframe = hdmi_out.newframe()


#whiLe(True):
start = time.time()
for j in range(numframes):
    #print(j)
    inframe = hdmi_in.readframe()
    outframe = hdmi_out.newframe()
    cv2.filter2D(inframe, -1, kernelF, dst=xFbuf1,borderType=cv2.BORDER_CO
NSTANT)
    difference = cv2.absdiff(reference_blur, xFbuf1)
    threshold = cv2.threshold(difference, 25, 255, cv2.THRESH_BINARY)[1]
    cv2.dilate(threshold, kernelVoid, dst=xFbuf2, iterations=1, borderType
=cv2.BORDER_CONSTANT)
    _, contours, hier = cv2.findContours(xFbuf2.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    for i in contours:
        if cv2.contourArea(i) < 4000:
            continue
        (x, y, w, h) = cv2.boundingRect(i)
        cv2.rectangle(inframe, (x, y), (x + w, y + h), (0, 0, 255), 2)
        hdmi_out.writeframe(inframe)
end = time.time()
print("Frames per second: " + str(numframes / (end - start)))
```

```
Run Fully Software\
Frames per second: 3.073178852200336
```

```
PIL.Image.fromarray(inframe)
```




```
print("Run Partial Hardware\")

inframe = hdmi_in.readframe()
reference_blur = cv2.GaussianBlur(inframe, (5, 5), 0)
kernelF = np.array([[0.0625,0.125,0.0625],[0.125,0.25,0.125],[0.0625,0.125
,0.0625]], np.float32)
kernelVoid = np.zeros(0)

xFbuf1 = mem_manager.cma_array((height,width),np.uint8)
xFbuf2 = mem_manager.cma_array((height,width),np.uint8)
numframes =10
outframe = hdmi_out.newframe()


#whiLe(True):
start = time.time()
for j in range(numframes):
    #print(j)
    inframe = hdmi_in.readframe()
    outframe = hdmi_out.newframe()
    xv2.filter2D(inframe, -1, kernelF, dst=xFbuf1,borderType=cv2.BORDER_CO
NSTANT)
    difference = cv2.absdiff(reference_blur, xFbuf1)
    threshold = cv2.threshold(difference, 25, 255, cv2.THRESH_BINARY)[1]
    cv2.dilate(threshold, kernelVoid, dst=xFbuf2, iterations=1, borderType
=cv2.BORDER_CONSTANT)
    _, contours, hier = cv2.findContours(xFbuf2.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    for i in contours:
        if cv2.contourArea(i) < 4000:
            continue
        (x, y, w, h) = cv2.boundingRect(i)
        cv2.rectangle(inframe, (x, y), (x + w, y + h), (0, 0, 255), 2)
        hdmi_out.writeframe(inframe)
```

```
end = time.time()
print("Frames per second: " + str(numframes / (end - start)))
```

Run Partial Hardware١
Frames per second: 8.373206324287107

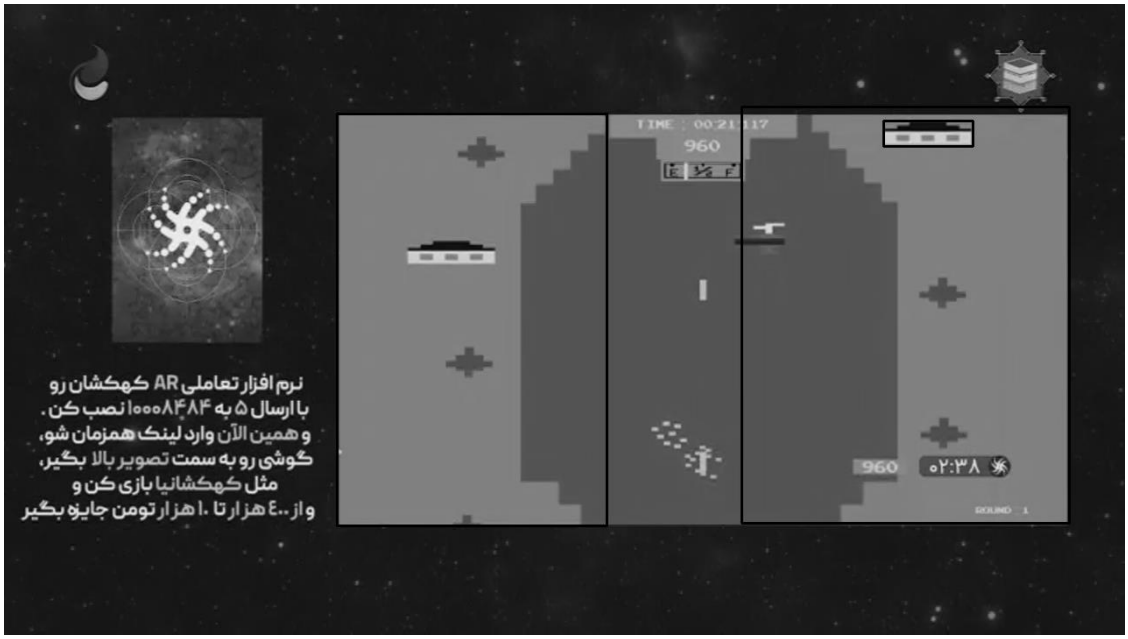
```
PIL.Image.fromarray(inframe)
```


```
print("Run Fully Software٢")

inframe = hdmi_in.readframe()
reference_blur = cv2.GaussianBlur(inframe, (5, 5), 0)
kernelF = np.array([[0.0625,0.125,0.0625],[0.125,0.25,0.125],[0.0625,0.125
,0.0625]], np.float32)
kernelVoid = np.zeros(0)

xFbuf1 = mem_manager.cma_array((height,width),np.uint8)
xFbuf2 = mem_manager.cma_array((height,width),np.uint8)
numframes =20
outframe = hdmi_out.newframe()

start = time.time()
for j in range(numframes):
    #print(j)
    inframe = hdmi_in.readframe()
    outframe = hdmi_out.newframe()
    cv2.filter2D(inframe, -1, kernelF, dst=xFbuf1,borderType=cv2.BORDER_CO
NSTANT)
    difference = cv2.absdiff(reference_blur, xFbuf1)
    threshold = cv2.threshold(difference, 25, 255, cv2.THRESH_BINARY)[1]
    cv2.dilate(threshold, kernelVoid, dst=xFbuf2, iterations=1, borderType
=cv2.BORDER_CONSTANT)
    _, contours, hier = cv2.findContours(xFbuf2.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    hdmi_out.writeframe(inframe)
```

```
end = time.time()

for i in contours:
    if cv2.contourArea(i) < 2000:
        continue
    (x, y, w, h) = cv2.boundingRect(i)
    cv2.rectangle(inframe, (x, y), (x + w, y + h), (0, 0, 255), 2)
hdmi_out.writeframe(inframe)
print("Frames per second: " + str(numframes / (end - start)))
```

Run Fully Software۲
Frames per second: 4.423886687096567

```
PIL.Image.fromarray(inframe)
```




```
print("Run Partial Hardware۲")

inframe = hdmi_in.readframe()
reference_blur = cv2.GaussianBlur(inframe, (5, 5), 0)
kernelF = np.array([[0.0625,0.125,0.0625],[0.125,0.25,0.125],[0.0625,0.125
,0.0625]], np.float32)
kernelVoid = np.zeros(0)

xFbuf1 = mem_manager.cma_array((height,width),np.uint8)
xFbuf2 = mem_manager.cma_array((height,width),np.uint8)
numframes =10
outframe = hdmi_out.newframe()


#whiLe(True):
start = time.time()
for j in range(numframes):
    #print(j)
    inframe = hdmi_in.readframe()
    outframe = hdmi_out.newframe()
```

```
    xv2.filter2D(inframe, -1, kernelF, dst=xFbuf1,borderType=cv2.BORDER_CO
NSTANT)
    difference = cv2.absdiff(reference_blur, xFbuf1)
    threshold = cv2.threshold(difference, 25, 255, cv2.THRESH_BINARY)[1]
    cv2.dilate(threshold, kernelVoid, dst=xFbuf2, iterations=1, borderType
=cv2.BORDER_CONSTANT)
    _, contours, hier = cv2.findContours(xFbuf2.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    hdmi_out.writeframe(inframe)
end = time.time()

for i in contours:
    if cv2.contourArea(i) < 2000:
        continue
    (x, y, w, h) = cv2.boundingRect(i)
    cv2.rectangle(inframe, (x, y), (x + w, y + h), (0, 0, 255), 2)
hdmi_out.writeframe(inframe)

print("Frames per second: " + str(numframes / (end - start)))
```

Run Partial Hardware۲
Frames per second: 12.801449382849437

```
PIL.Image.fromarray(inframe)
```


```
hdmi_out.close()
hdmi_in.close()
```

در اجراهای شماره ۱ ترسیم کانتور نیز به صورت بلادرنگ لحاظ شده است، اما در اجراهای شماره ۲ به دلیل یکسان نبودن کانتورها و تفاوت زمان ترسیم این مورد تنها به فریم آخر اعمال

می‌گردد لذا در زمان اجرای حلقه تاثیری نخواهد داشت. میزان سرعت بخشی بدست آمده عبارتست از :

No	1	2
Hardware	8.373206324287107	12.801449382849437
Software	3.073178852200336	4.423886687096567
Speed Up	٢,٧٢٤٦٠٧٥٦	٢,٨٩٣٧١٠٩٦

به طور مشخص دلیل کمتر بودن افزایش سرعت نسبت به حالت قبلی اجرای بخش‌هایی ار الگوریتم به صورت نرم‌افزاری در کد می‌باشد.

ارتباط با موتورها :

جهت ارتباط با موتورها از یک رابط خاص که در PYNQ برای ارتباط با سخت‌افزار خارجی تعبیه شده است استفاده می‌گردد. این رابط مبتنی بر Microblaze بوده و کد C را مستقیما به عنوان ورودی دریافت کرده و پس از اجرا عملیات مورد نظر را انجام می‌دهد. از کد زیر به عنوان درایور موتورها استفاده می‌گردد :

```
//microblaze base.ARDUINO


#include "xio_switch.h"
#include "gpio.h"
#include "timer.h"

#define DEFAULT_PERIOD 625998
#define DEFAULT_DUTY 312998

#define PWM_A_PIN 3
```

```
#define PWM_B_PIN 11
#define DIR_A_PIN 12
#define DIR_B_PIN 14

typedef enum motor}
MOTOR_A = 0,
MOTOR_B = 1,
{motor_e;

static unsigned int pol_a = 0, pol_b = 0;
static unsigned int dir_a = 0, dir_b = 0;
static unsigned int duty_a = 50, duty_b = 50;

static timer timer_a;
static timer timer_b;
static gpio gpio_a;
static gpio gpio_b;

unsigned int init_ardumoto  }()
  timer_a = timer_open_device;(·)
  timer_b = timer_open_device;(ბ)
  set_pin(PWM_A_PIN, PWM0);
  set_pin(PWM_B_PIN, PWM5);
  gpio_a = gpio_open(DIR_A_PIN);
  gpio_b = gpio_open(DIR_B_PIN);
```

```
    gpio_set_direction(gpio_a, GPIO_OUT);
    gpio_set_direction(gpio_b, GPIO_OUT);
    return 0;
{


void configure_polar(unsigned int motor, unsigned int polarity)}
    if (motor == MOTOR_A)}
        pol_a = polarity;
{   else if (motor == MOTOR_B}(
        pol_b = polarity;
{
{


void set_direction(unsigned int motor, unsigned int direction)}
    if (motor == MOTOR_A)}
        dir_a = (direction)? pol_a : !pol_a;
{
    else if (motor == MOTOR_B)}
        dir_b = (direction)? pol_b : !pol_b;
{
{


void set_speed(unsigned int motor, unsigned int speed)}
```

```
   if (motor == MOTOR_A)}
      duty_a = speed;
{    else if (motor == MOTOR_B} (
      duty_b = speed;
{
{

void run(unsigned int motor)}
   if (motor == MOTOR_A)}
      gpio_write(gpio_a, dir_a);
      timer_pwm_generate(timer_a, DEFAULT_PERIOD ,
                  duty_a*DEFAULT_PERIOD/100;(
{    else if(motor == MOTOR_B} (
      gpio_write(gpio_b, dir_b);
      timer_pwm_generate(timer_b, DEFAULT_PERIOD ,
                  duty_b*DEFAULT_PERIOD/100;(
{
{

void stop(unsigned int motor)}
   if (motor == MOTOR_A)}
      timer_pwm_stop(timer_a);
```

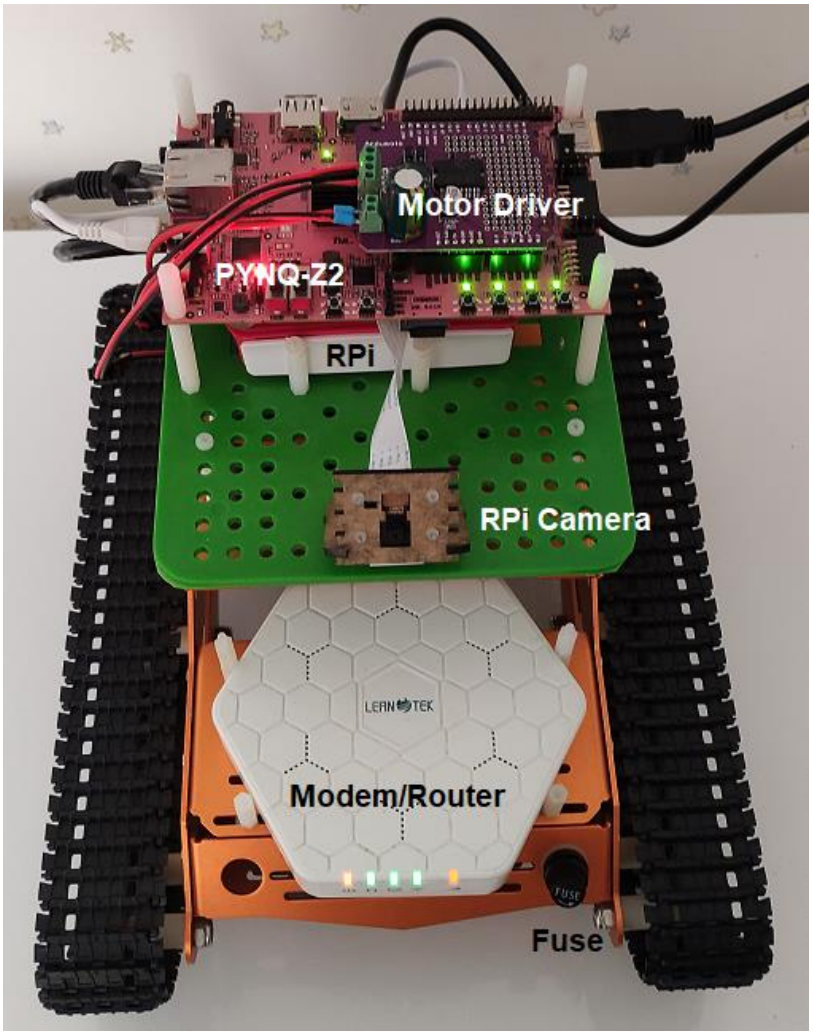
```
{   else if (motor == MOTOR_B}(
      timer_pwm_stop(timer_b);
{
{
```

در این کد می‌توان با تنظیم مقدار PWM هر موتور سرعت آن را به صورت جداگانه تنظیم نمود، همچنین امکان تغییر جهت موتور نیز در این کد وجود دارد.

با افزودن این بخش به کد قبلی می‌توان بنا بر موقعیت کانتور تشخیص داده شده حرکت ربات را به سمت آن تنظیم نمود.

سخت‌افزار سیستم:

در تصاویر زیر سخت‌افزار مورد استفاده پروژه جهت اجرای کد آورده شده است. هر قسمت این سخت‌افزار به صورت مختصر توضیح داده می‌شود :

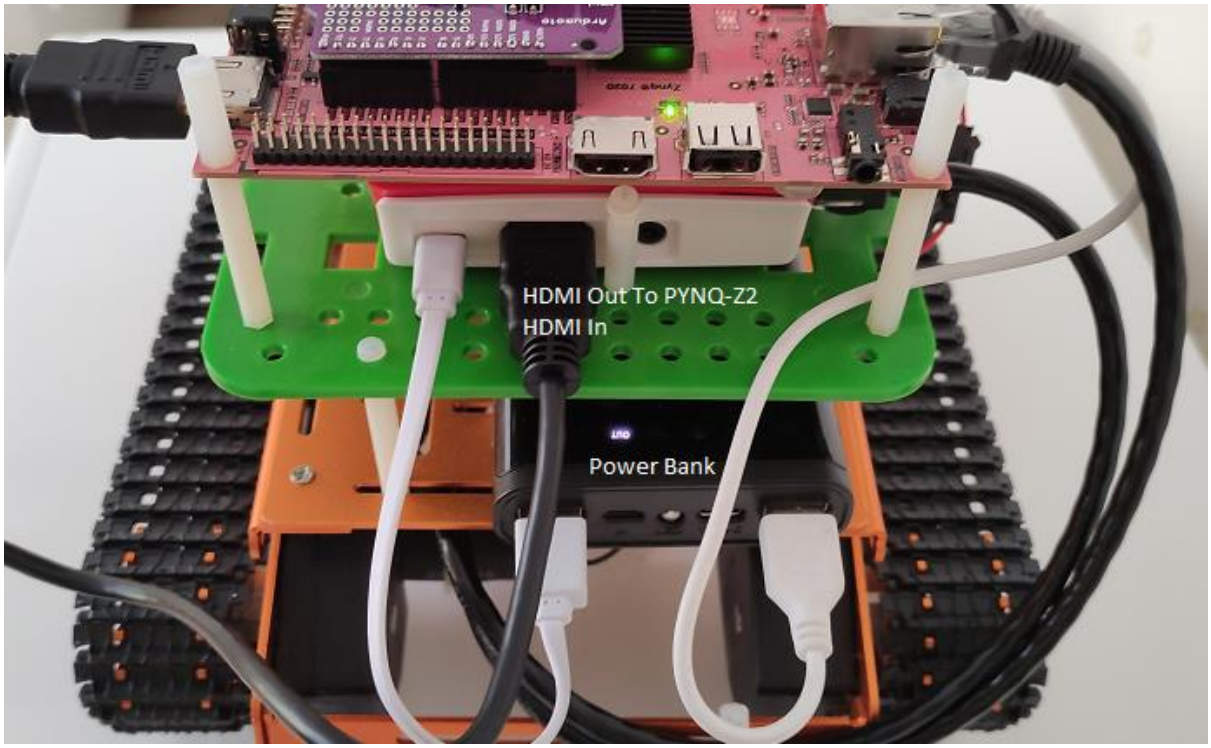


جهت تغذیه مودم و موتورها از یک باتری Lead Acid ۱۲ ولت با ظرفیت ۱.۲ آمپر ساعت استفاده شده است و از یک فیوز پیچی در جلوی ربات برای حفاظت و همچنین قطع راحت جریان استفاده شده است.

مودم جلوی دستگاه جهت اتصال به اینترنت برای دانلود آپدیت‌ها، نصب نرم‌افزارهای مورد نیاز برد Raspberry Pi و PYNQ مورد استفاده قرار می‌گیرد. همچنین از طریق رابط WiFi مودم اتصال به ربات برقرار می‌شود.

دوربین RPi[5] به اتصال به RPi از طریق رابط CSI[6] تصویر دریافتی را به برد RPi انتقال داده و با استفاده از درایور Libcamera و برنامه Libcamera-hello به صورت HDMI تبدیل می‌گردد. به دلیل عدم پشتیبانی از Anti-Flicker در این درایور برای دوربین استفاده شده، تصویر خروجی خصوصا در نور مصنوعی دارای چشمک می‌باشد که این مساله باعث کاهش کارایی الگوریتم پردازش تصویر می‌گردد. در در تصویر محل قرارگیری برد RPi در باکس مربوطه مشخص شده است. مدل مورد استفاده RPi 3B+ می‌باشد. این برد با استفاده از یک رابط شبکه به مودم متصل می‌باشد.

در قسمت بالای ربات برد PYNQ و درایور موتورها قرار گرفته‌اند. برد PYNQ با استفاده از یک رابط شبکه به مودم متصل می‌باشد و همچنین ورودی HDMI برد به خروجی HDMI RPi متصل شده است.



[5] - Raspberry Pi
[6] - Camera Serial Interface

جهت تغذیه برد RPi و PYNQ از یک پاوربانک با ظرفیت ۲۰۰۰۰ میلی آمپرساعت (اسمی) استفاده شده است. طی تست های انجام شده این پاور بانک در حدود ٤ الی ٥ ساعت توانایی ارائه توان به دو برد یاد شده را دارد.

در قسمت زیرین ربات یک باتری Lead Acid ۱۲ ولت با ظرفیت ۱.۲ آمپر ساعت قرار داده شده است. همانگونه که اشاره گردید این باتری جهت تغذیه مودم و موتورها استفاده شده است. در حالت اتصال به اینترنت با سیم کارت نسل ٤ در شرایط آنتن‌دهی مناسب، استفاده از موتورها در تست‌ها و همچنین اتصال WiFi به کامپیوتر این باتری در حدود ۱ ساعت و ۳۰ دقیقه توانایی تغذیه سیستم را دارد. به همین دلیل برای تست از آنجا که این زمان کافی نمی‌باشد یک فیش آداپتوری در نظر گرفته شده است که می‌توان این بخش را با یک آداپتور ۱۲ ولت ۲ آمپری تغذیه نمود.

نتیجه‌گیری :

استفاده از پلتفرم PYNQ می‌تواند کار با تراشه‌های برنامه‌پذیر سری Zynq, Zynq Ultrascale را به شدت ساده نماید اما در عین سادگی برخی مشکلات در خصوص به کارگیری این روش وجود دارد. از جمله این موارد می‌توان به نبود امکان عیب‌یابی برنامه در حال اجرا به طور مناسب و همچنین عدم امکان استفاده از تمامی قابلیت‌های بخش PL به دلیل ذات اجرای پشت سرهم قسمت PS نام برد. دلیل عمده این مساله نیاز اکثر برنامه‌های ایجاد شده به بارگذاری لاجیک و اطلاعات از قسمت PS می‌باشد. همچنین در خصوص رابط‌های بین PS , PL و همچنین

نتایج بارگذاری Partial (از نسخه ۲.٤ به بعد) تا نسخه ۲.۷ روشی برای اطلاع از اشغال بودن رابط‌ها و همچنین سهولت جایگذاری فایل‌های Bit ایجاد نشده است که احتمالا در نسخه ۲.۸ اضافه خواهد شد. در بسیاری از موارد نیز Overlay های مورد نیاز برای الگوریتم برای اجرای بهینه بهتر است مجددا توسط طراح سیستم ایجاد شود که این مساله به دانش سخت‌افزاری مجددا نیاز خواهد داشت. البته بایستی توجه داشت که برای این منظور از روش‌های مبتنی بر HLS, Model Based و RTL می‌توان استفاده نمود همچنین تعداد بسیار زیادی پیاده‌سازی‌های مختلف برای نمونه وجود دارد. در سال‌های اخیر استفاده از PYNQ برای پیاده‌سازی الگوریتم‌های مختلف خصوصا موارد مرتبط با هوش‌مصنوعی و یادگیری ماشین به شدت افزایش یافته است و برای این منظور شرکت AMD یک کتابخانه به همراه Overlay های مربوطه را به نام FINN ایجاد نموده است که کار با این موارد را بسیار ساده می‌سازد. در ادامه این پروژه سعی می‌گردد با استفاده از این روش یک سیستم هوش‌مصنوعی بر پایه تشخیص تصاویر بر روی سیستم ایجاد شده و مورد تست قرار گیرد.