Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Tutorial Proyecto 1

Taller de Sistemas Embebidos

Estudiante:

Tobías Fonseca Cruz

Profesor:

Dr.-Ing. Johan Carvajal Godínez

23 de septiembre de 2024

Índice

1. Características computador host

A continuación, se detallan las características del computador host donde se desarrolló el proyecto:

- Sistema operativo: Ubuntu 22.04 LTS 64-bit
- Procesador: 13th Gen Intel(R) Core(TM) i5-13420H
- Memoria RAM: 16 GB DDR5 4800MHz
- Almacenamiento: 512 GB (150 GB para Ubuntu)

2. Toolkits

A continuación se listan los toolkits y se da una descripción de cada uno:

2.1. OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto enfocada en aplicaciones de visión por computadora y aprendizaje automático. Fue desarrollada inicialmente por Intel y ahora es mantenida por una comunidad global. OpenCV proporciona herramientas para el procesamiento de imágenes, análisis de video, reconocimiento de objetos y otras tareas relacionadas con la visión artificial.

Principales Funcionalidades de OpenCV

- **Procesamiento de Imágenes:** Ofrece una amplia gama de operaciones, como filtros de suavizado, detección de bordes, transformaciones geométricas y corrección de color. Facilita el preprocesamiento de imágenes para su análisis posterior.
- **Detección y Reconocimiento de Objetos:** Proporciona algoritmos para la detección de caras, reconocimiento de texto (OCR), y segmentación de objetos, utilizando tanto métodos tradicionales como redes neuronales profundas.
- **Análisis de Video:** Soporta la captura y el procesamiento de secuencias de video en tiempo real, con funcionalidades como seguimiento de objetos, detección de movimiento y estimación de flujo óptico.
- **Algoritmos de Aprendizaje Automático:** Incluye una implementación de varios algoritmos de aprendizaje automático, como SVM, K-means y redes neuronales, para clasificar y analizar datos.

- **Compatibilidad Multiplataforma:** Es compatible con diversos lenguajes de programación, como C++, Python y Java, y puede ejecutarse en múltiples sistemas operativos, incluyendo Windows, Linux y macOS, así como en dispositivos móviles.

Ventajas de OpenCV

- **Eficiencia y Velocidad:** Está optimizada para aprovechar la aceleración por hardware en CPUs y GPUs, lo que permite el procesamiento de imágenes y videos en tiempo real.
- **Amplia Comunidad y Soporte:** Cuenta con una gran comunidad de desarrolladores y extensa documentación, lo que facilita la resolución de problemas y el desarrollo de proyectos complejos.
- **Integración con Bibliotecas Externas:** Se integra fácilmente con otras bibliotecas y frameworks, como TensorFlow, PyTorch y OpenVINO, permitiendo el uso combinado de técnicas de visión por computadora y aprendizaje profundo.

2.2. OpenVINO

OpenVINO (Open Visual Inference and Neural Network Optimization) es un *toolkit* de código abierto desarrollado por Intel para optimizar y ejecutar modelos de aprendizaje profundo y visión por computadora en diversos dispositivos, como CPUs, GPUs, FPGAs y VPUs (Vision Processing Units). Su principal objetivo es mejorar el rendimiento de la inferencia de modelos de redes neuronales y facilitar su implementación en aplicaciones de inteligencia artificial.

Principales Componentes de OpenVINO

- **Modelo de Representación Intermedia (IR):** Convierte modelos de *frameworks* populares (TensorFlow, PyTorch, ONNX, etc.) a un formato intermedio (.xml y .bin) que es optimizado para ejecutarse en diversos dispositivos de hardware.
- **Model Optimizer:** Realiza optimizaciones en los modelos, como la eliminación de capas redundantes y la reducción de precisión (por ejemplo, de FP32 a INT8) para mejorar la velocidad de inferencia y reducir el uso de recursos.
- **Inference Engine:** Motor de inferencia que ejecuta modelos en diferentes dispositivos, proporcionando una API unificada para manejar la inferencia en CPUs, GPUs, VPUs y FPGAs. Soporta ejecución heterogénea, permitiendo que partes del modelo se ejecuten en diferentes dispositivos.

- **Model Zoo:** Conjunto de modelos preentrenados para tareas comunes de visión por computadora (detección de objetos, clasificación de imágenes, reconocimiento facial) que se pueden usar directamente o como punto de partida para aplicaciones específicas.
- **Herramientas Complementarias:**
  - **Benchmark Tool:** Mide el rendimiento de los modelos en diferentes dispositivos.
  - **Calibration Tool:** Facilita la conversión de modelos a precisiones menores como INT8, optimizando aún más el rendimiento.
  - **Deployment Manager:** Ayuda a desplegar modelos en diferentes dispositivos de manera eficiente.

Ventajas de OpenVINO

- **Optimización de Rendimiento:** Aumenta la velocidad de inferencia y reduce el consumo de recursos, maximizando el uso del hardware disponible.
- **Compatibilidad Multiplataforma:** Permite la implementación en una amplia gama de dispositivos, desde PCs hasta dispositivos IoT.
- **Flexibilidad y Escalabilidad:** Facilita la implementación de modelos en entornos *edge* y *cloud*, soportando tanto dispositivos de alta como de baja potencia.

2.3. Yocto Project

Yocto Project es un proyecto de código abierto que proporciona un marco para crear sistemas operativos personalizados basados en Linux para dispositivos embebidos. Su principal objetivo es facilitar el desarrollo de distribuciones de Linux adaptadas a las necesidades específicas de hardware y software de los desarrolladores de dispositivos embebidos.

Principales Componentes de Yocto Project

- **BitBake:** Es la herramienta de construcción que permite definir cómo se construyen las imágenes y los paquetes. Utiliza recetas (recipes) para describir las fuentes, dependencias y pasos de compilación.
- **OpenEmbedded:** Es un sistema de metadatos que proporciona una base para la construcción de software en plataformas embebidas. Contiene una amplia variedad de recetas y capas (layers) que facilitan la integración de diversas aplicaciones y bibliotecas.

- **Poky:** Es la referencia de distribución de Yocto Project, que incluye las herramientas de construcción y un conjunto básico de recetas y paquetes. Sirve como punto de partida para crear imágenes personalizadas.
- **Layers:** Permiten organizar las recetas y paquetes en capas modulares, lo que facilita la personalización y reutilización del código. Cada capa puede contener paquetes, configuraciones y recetas específicas.
- **SDK (Software Development Kit):** Genera un kit de desarrollo para facilitar el desarrollo de aplicaciones para la plataforma embebida. Incluye bibliotecas, herramientas y ejemplos para ayudar a los desarrolladores a comenzar rápidamente.

Ventajas de Yocto Project

- **Personalización:** Permite crear distribuciones de Linux altamente personalizadas, adaptadas a las necesidades específicas del hardware y software.
- **Compatibilidad Multiplataforma:** Soporta una amplia gama de arquitecturas de hardware, desde microcontroladores hasta sistemas embebidos más complejos.
- **Gestión de Dependencias:** Facilita la gestión de dependencias de software, asegurando que todas las bibliotecas y herramientas necesarias se incluyan en la imagen final.
- **Comunidad Activa:** Cuenta con una comunidad activa que contribuye constantemente con nuevas recetas y mejoras, lo que permite mantenerse actualizado con las últimas tecnologías y herramientas.

2.4. VirtualBox

VirtualBox es un software de virtualización de código abierto desarrollado por Oracle que permite a los usuarios ejecutar múltiples sistemas operativos en una sola máquina física. Es compatible con una amplia variedad de sistemas operativos, incluidos Windows, Linux, macOS y Solaris, lo que lo convierte en una herramienta versátil para desarrolladores, administradores de sistemas y usuarios en general.

Principales Funcionalidades de VirtualBox

- **Máquinas Virtuales (VM):** Permite crear y gestionar múltiples máquinas virtuales en una sola computadora, cada una de las cuales puede tener su propio sistema operativo y configuraciones de hardware.
- **Instantáneas (Snapshots):** Ofrece la capacidad de tomar instantáneas de una máquina virtual en un momento dado, permitiendo a los usuarios revertir el

estado de la VM a una versión anterior fácilmente.

- **Compatibilidad con Hardware Virtualizado:** Soporta la virtualización de hardware, lo que permite a las máquinas virtuales aprovechar características como la aceleración por hardware y el soporte para dispositivos USB.
- **Integración con el Host:** Proporciona herramientas de integración que permiten compartir archivos y portapapeles entre el sistema operativo host y las máquinas virtuales, mejorando la experiencia del usuario.
- **Redes Virtuales:** Permite configurar diversas opciones de red para las máquinas virtuales, incluyendo NAT, red interna y red puenteada, facilitando la comunicación entre VM y con el mundo exterior.

Ventajas de VirtualBox

- **Código Abierto y Gratuito:** Al ser un proyecto de código abierto, VirtualBox es accesible y gratuito, lo que permite a los usuarios personalizar y contribuir al software.
- **Fácil de Usar:** La interfaz gráfica de usuario es intuitiva y fácil de navegar, lo que permite a los usuarios configurar y gestionar máquinas virtuales sin complicaciones.
- **Amplia Compatibilidad:** Es compatible con múltiples sistemas operativos y hardware, lo que permite a los usuarios ejecutar casi cualquier sistema operativo en su computadora.
- **Soporte de Comunidad:** Cuenta con una gran comunidad de usuarios y desarrolladores que ofrecen soporte y documentación, facilitando la resolución de problemas y la implementación de nuevas características.

2.5. Micromamba

Micromamba es un gestor de paquetes y entornos de código abierto diseñado para la instalación y gestión de software en entornos de desarrollo, especialmente en el ecosistema de Python y otros lenguajes. Es una versión más ligera y rápida de Mamba, que a su vez es una alternativa de alto rendimiento a Conda.

Principales Funcionalidades de Micromamba

- **Instalación Rápida:** Permite la instalación de paquetes y la creación de entornos de manera eficiente y rápida, optimizando el proceso de resolución de dependencias.

- **Gestión de Entornos:** Facilita la creación, activación y desactivación de entornos virtuales, permitiendo a los desarrolladores trabajar en proyectos con diferentes dependencias sin conflictos.
- **Compatibilidad:** Es compatible con el formato de paquetes de Conda, lo que permite a los usuarios instalar y gestionar paquetes de Conda de manera sencilla.
- **Ligero y Portátil:** Micromamba está diseñado para ser ligero y fácil de utilizar, lo que permite su integración en proyectos y entornos con restricciones de recursos.
- **Sin Dependencias Adicionales:** Se puede utilizar sin necesidad de instalar todo el ecosistema de Conda, lo que lo hace ideal para situaciones donde se requiere una solución rápida y sin complicaciones.

Ventajas de Micromamba

- **Rendimiento Mejorado:** Gracias a su diseño optimizado, Micromamba es significativamente más rápido en comparación con otros gestores de paquetes, lo que mejora la productividad del desarrollador.
- **Simplicidad:** La interfaz de línea de comandos es sencilla y directa, facilitando la gestión de paquetes y entornos para usuarios de todos los niveles.
- **Ideal para CI/CD:** Su ligereza y rapidez lo hacen especialmente adecuado para pipelines de integración continua y despliegue continuo (CI/CD).
- **Código Abierto:** Al ser un proyecto de código abierto, Micromamba es accesible para la comunidad, lo que permite contribuciones y mejoras continuas.

3. Flujo de trabajo

A continuación, se describe el proceso para la implementación de la aplicación, la creación de la imagen y la inicialización de la máquina virtual con los comandos y pasos correspondientes para cada etapa.

3.1. Aplicación

Detección de rocas

La aplicación inicial consiste en un algoritmo para la detección de rocas en la Luna y/ Marte, para ser usada por el sistema de navegación de un rover. Utilizando el dataset de Roboflow [1] se entrena el modelo Yolov8n.pt y se exporta a ONNX. Mediante la librería de Ultralytics, se realiza la detección de rocas con un video de prueba tal como muestra la Figura 1.

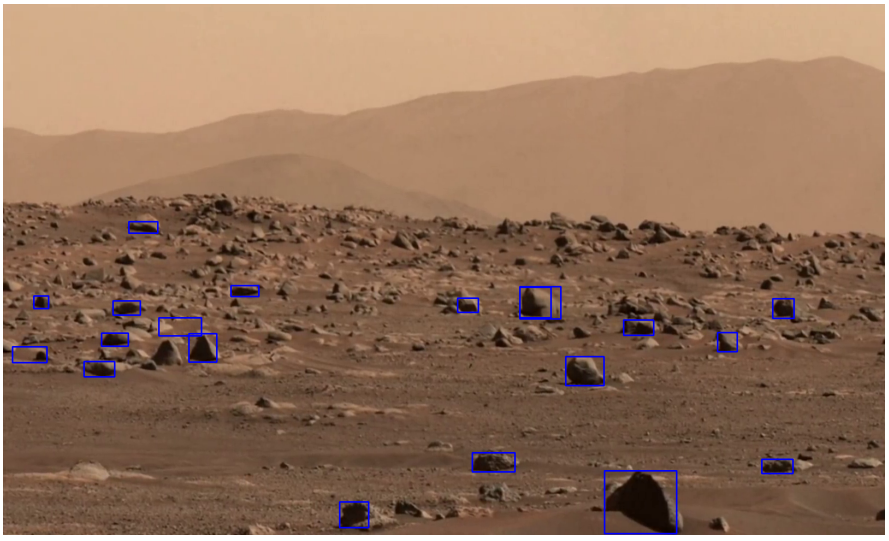


Figura 1: Detección de rocas con modelo entrenado

Es posible observar que se detectan bien ciertas rocas y el modelo funciona, claramente el video muestra muchas rocas a la vez por lo que resulta complicado poder determinar todas las presentes. Al momento de intentar generar la imagen, los metas necesarios para la creación de la imagen con Yocto Project presentaron problemas. Por lo que, dada la cercanía de la entrega del proyecto se decide utilizar una aplicación más genérica para poder realizar el ejercicio de la creación de la imagen.

Detección de autos

Tomando como base códigos encontrados [2], se realiza una detección de automóviles en carretera. Utilice el siguiente comando para clonar el repositorio con los archivos necesarios:

```
$ git clone https://github.com/Tobiasfonseca/p1_embebidos.git
```

Dentro del repositorio se encuentran dos script de Python:

- cars.py: este programa realiza la detección de los carros siguiendo el pipe de la Figura 2.

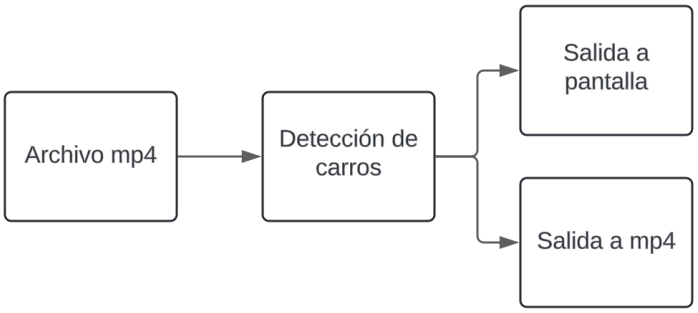


Figura 2: Pipe del programa cars

  Como entrada se tiene un archivo mp4 el cual es definido por el usuario y se toma un frame a la vez para ser procesado. El frame pasa por la detección y se muestra en la pantalla y a la vez se almacena en otro archivo mp4 llamado output.mp4. El modelo utilizado corresponde a "ssd_mobilenet_v2_fp16" el cual se puede obtener desde el repositorio de OpenVINO [3].El motor de inferencia OpenVINO procesa cada frame y al superar cierto umbral en la detección se dibuja el bounding box sobre la detección hallada. El frame se retorna y es transmitido a la pantalla y guardado en un nuevo archivo mp4 de salida.

- video_player.py: el programa se encarga de reproducir el archivo mp4 de salida generado por cars.py.

Las dependencias para estos programas son:

- cv2
- numpy
- openvino
- pathlib

Para la ejecución de la aplicación en el computador host se utiliza un ambiente de micromamba, para la instalación se utiliza el siguiente comando:

```
$ "${SHELL}" <(curl -L micro.mamba.pm/install.sh)
```

Para que tome efecto se cierra la terminal y se abre otra nueva o se reinicia. Luego, se crea un ambiente:

```
$ micromamba create -n P1
```

Si fue satisfactoria la instalación y la creación del ambiente, la terminal debe verse de la siguiente manera:

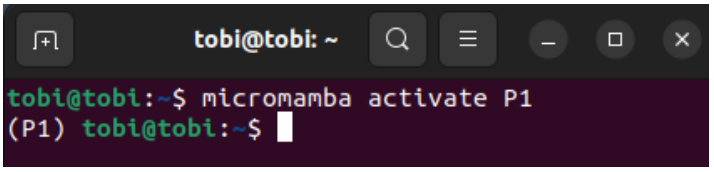


Figura 3: Terminal con ambiente de micromamba activado

Una vez activado el ambiente, se instalan los siguientes paquetes:

```
$ micromamba install numpy
$ micromamba install -c conda-forge opencv
$ micromamba install -c conda-forge ocl-icd-system
$ micromamba install openvino=2024.3
```

Luego, se ubica la terminal dentro de la carpeta clonada con los archivos y se ejecuta:

```
$ python3 cars.py
```

El programa solicita el nombre del video de prueba, al ingresarlo se abre una ventana mostrando la detección de carros como se logra observar en la Figura 5.

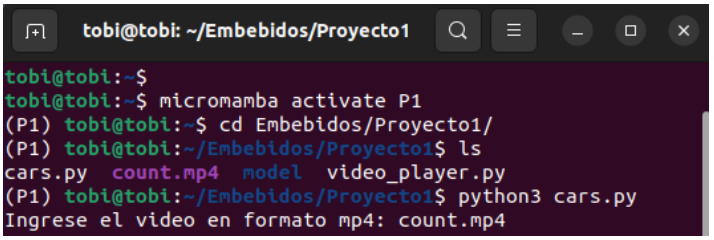


Figura 4: Terminal con comandos para la ejecución del script

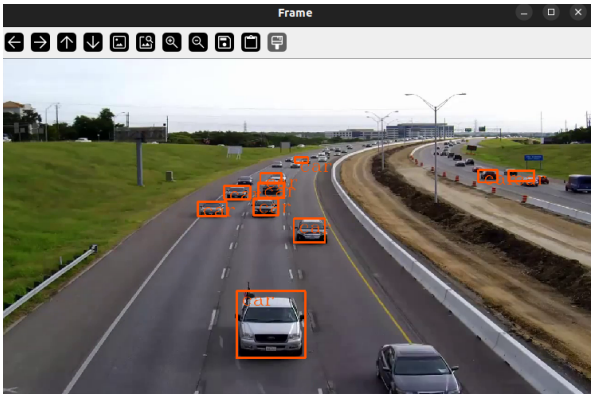


Figura 5: Salida de detección de carros

3.2. Creación de imagen

3.2.1. Requisitos

Para la creación de la imagen con Yocto Project es importante cumplir con los siguientes requisitos [4]:

- Al menos 90 GB de almacenamiento libre
- 8 GB de RAM
- Distribución de Linux soportada
- Git 1.8.3.1 o superior
- tar 1.28 o superior
- Python 3.8.0 o superior
- gcc 8.0 o superior
- GNU make 4.0 o superior

3.2.2. Instalación

Si se cumplen los requisitos, se continúa con la construcción de paquetes esenciales:

```
$ sudo apt install gawk wget git diffstat unzip texinfo gcc build-essential chrpath socat cpio
    python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git
    python3-jinja2 libegl1-mesa libsdl1.2-dev python3-subunit mesa-common-dev zstd
    liblz4-tool file locales libacl1
$ sudo locale-gen en_US.UTF-8
```

Luego, se clona el repositorio de poky y se cambia al branch de Nanbield con los siguientes comandos:

```
$ git clone git://git.yoctoproject.org/poky
$ cd poky
$ git checkout -t origin/nanbield -b my-nanbield
$ git pull
```

Luego, se inicializa el ambiente de construcción para poder definir las características deseadas en la imagen:

```
$ source oe-init-build-env
```

El comando ubicará la terminal en la carpeta *build*, donde se encuentran los archivos de configuración para la imagen.

3.2.3. Recetas

Con la terminal ubicada en el directorio de poky, se clona el repositorio de meta-openembedded y se cambia al branch Nanbield:

```
$ git clone https://git.openembedded.org/meta-openembedded
$ cd meta-openembedded
$ git checkout -t origin/nanbield -b my-nanbield
```

Nuevamente en el directorio de poky, se clona meta-intel:

```
$ git clone https://git.yoctoproject.org/git/meta-intel
$ cd meta-intel
$ git checkout -t origin/nanbield -b my-nanbield
```

Luego, se genera un meta-layer para agregar los archivos de python y videos dentro de la imagen, con la terminal ubicada en *build*:

```
$ bitbake-layers create-layer ../meta-mylayer
$ cd ..
$ cd meta-mylayer
$ cd recipes-example
$ mkdir file-boost
$ cd file-boost
$ mkdir files
```

Utilizando un editor, se genera el archivo file-boost.bb, el cual es una receta para agregar los archivos del proyecto:

```
$ nano file-boost.bb
```

Dentro del archivo se copia lo siguiente:

```
SUMMARY = "Recipe to include project files"
LICENSE = "CLOSED"

SRC_URI = "file://Proyecto1"
S = "${WORKDIR}"

do_install () {
     install -d ${D}${bindir}/Proyecto1
     cp -r Proyecto1/* ${D}${bindir}/Proyecto1
}
```

Dentro de la carpeta file, debe agregarse la carpeta del proyecto.

Seguidamente, se modifica el archivo *bblayers.conf* ubicado en build/conf:

```
# POKY_BBLAYERS_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /media/tobi/Yocto/poky/meta \
  /media/tobi/Yocto/poky/meta-poky \
  /media/tobi/Yocto/poky/meta-yocto-bsp \
  /media/tobi/Yocto/poky/meta-openembedded/meta-oe \
  /media/tobi/Yocto/poky/meta-openembedded/meta-python \
  /media/tobi/Yocto/poky/meta-openembedded/meta-multimedia \
  /media/tobi/Yocto/poky/meta-intel \
  /media/tobi/Yocto/poky/meta-mylayer \
  "
```

En la misma ubicación, se abre el archivo *local.conf* y se agregan las siguientes líneas:

```
IMAGE_FSTYPES += "wic.vmdk"

# Enable building inference engine python API.
# This requires meta-python layer to be included in bblayers.conf.
PACKAGECONFIG:append:pn-openvino-inference-engine = " python3"
# Include OpenVINO Python API package in the target image.
LICENSE_FLAGS_ACCEPTED += "commercial"
```

```
IMAGE_INSTALL:append = " openssh \
        python3 \
        python3-pip \
        python3-numpy \
        opencv \
        openvino-inference-engine \
        openvino-model-optimizer \
        openvino-inference-engine-python3 \
        gstreamer1.0 \
        gstreamer1.0-plugins-base \
        gstreamer1.0-plugins-good \
        gstreamer1.0-plugins-bad \
        gstreamer1.0-plugins-ugly \
        gstreamer1.0-libav \
        gstreamer1.0-rtsp-server \
        gstreamer1.0-vaapi \
         file -boost"
```

Retornando al directorio de poky,se realiza un cambio en la receta de opencv:

```
$ cd meta-openembedded/meta-oe/recipes-support/opencv/
$ nano opencv_4.8.0.bb
```

Se busca la línea que contiene *PACKAGECONFIG[dnn]* y se activa *opencv_dnn*:

```
PACKAGECONFIG[dnn] = "-DBUILD_opencv_dnn=ON -DPROTOBUF_UPDATE_FILES=ON
-DBUILD_PROTOBUF=OFF -DCMAKE_CXX_STANDARD=17,
-DBUILD_opencv_dnn=ON,protobuf protobuf-native,"
```

Finalmente, en el directorio *build* se ejecuta bitbake y se genera la imagen:

```
$ bitbake core-image-x11
```

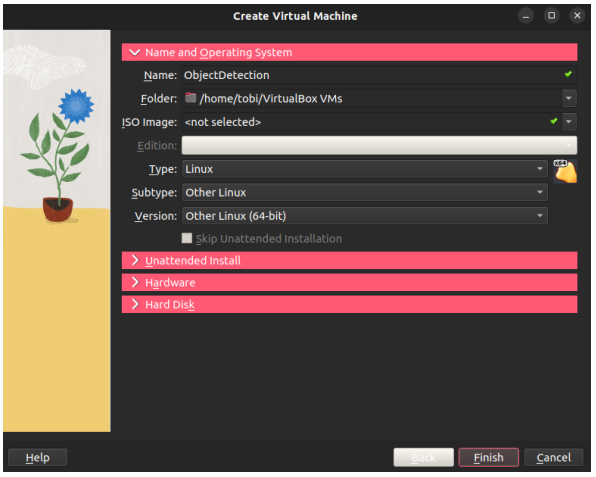
Es común que fallen ciertas tareas de instalación con código de error 1, generalmente se debe a que los procesos en ejecución consumen muchos recursos en la construcción y el proceso de bitbake los detiene para evitar posibles reinicios de la computadora. Basta con volver a correr el comando las veces necesarias hasta que finalicen las tareas.

3.3. Máquina virtual en VirtualBox

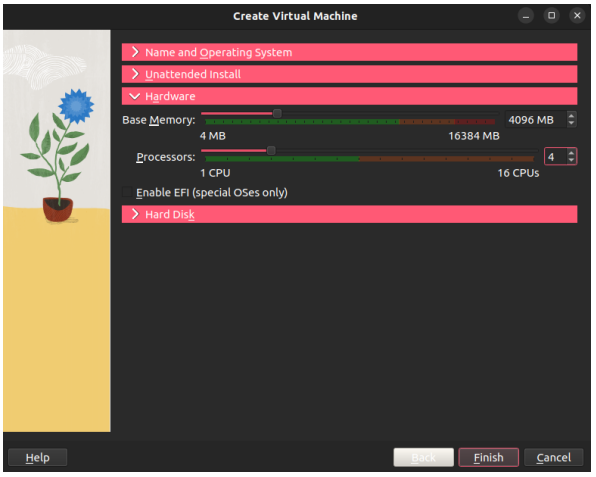
Si no se cuenta con VirtualBox, se puede instalar desde el sitio web seleccionando la verión correspondiente al sistema operativo o mediante la consola:

```
$ sudo apt-get install virtualbox
```

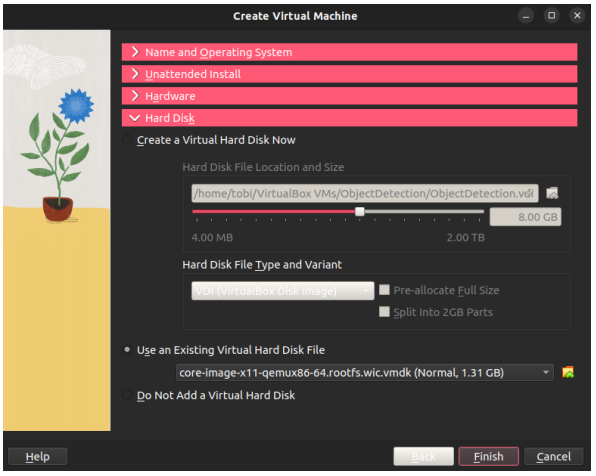
Al abrir el programa, se da click en **New** y se llena la siguiente información en **Name and Operating System**:



La pestaña **Unattended install** se omite y se continúa con **Hardware** donde se selecciona la memoria RAM y CPUs disponibles para la máquina:



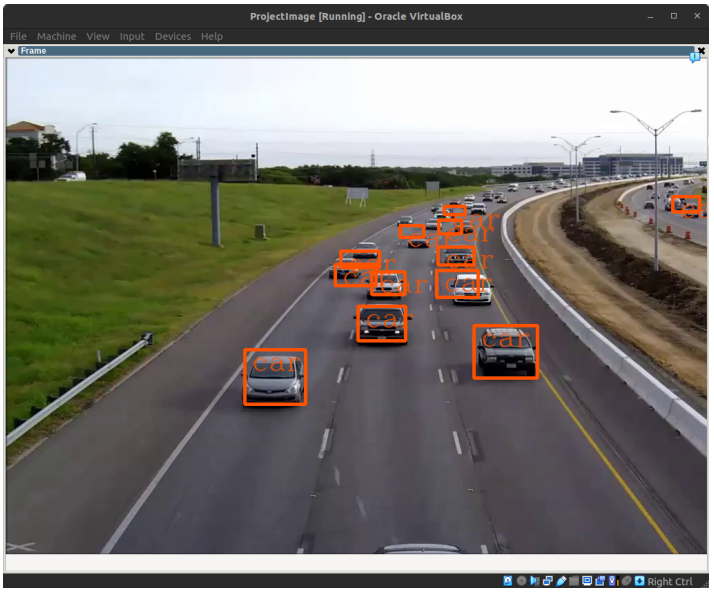
Por último, en **Hard Disk** se selecciona *Use an Existing Virtual Hard Disk File* y se busca la imagen generada por poky en el directorio poky/build/tmp/images/qemux86-64 y se selecciona core-image-x11-qemux86-64.rootfs.wic.vmdk



Se da terminar a la configuración y se inicia la máquina virtual. Para ejecutar la aplicación es necesario utilizar los siguientes comandos:

```
$ cd ..
$ cd ..
$ cd usr/bin/Proyecto1/
$ cd python3 cars.py
```

Donde se logra ejecutar la aplicación satisfactoriamente:



4. Dependencias

A continuación se resumen todas las dependencias necesarias tanto para la aplicación como para la imagen:

- OpenCV
- OpenVINO
- Python3
- Numpy
- Pathlib
- GStreamer
- X11: gestor de ventanas
- Archivos del proyecto

En cuanto a las recetas:

- meta-oe: capa que contiene las recetas para OpenCV
- meta-python: capa que contiene las recetas para Python3, Numpy, Pathlib
- meta-intel: capa que contiene las recetas para OpenVINO
- meta-multimedia: capa que contiene las recetas para para GStreamer y dependencias de OpenCV
- meta-mylayer: capa que contiene las recetas para la instalación de los archivos fuente de la aplicación

Se utiliza una imagen *core-image-x11* ya que esta incluye el gestor de ventanas X11.

Referencias

[1] Roboflow. Nasa rockyard object detection dataset (v5, 2024-08-12 7:31am) by nasarockyard. [Online]. Available: https://universe.roboflow.com/nasarockyard/nasa-rockyard/dataset/5

[2] Intel Corporation. Object detection with openvino toolkit. [Online]. Available: https://colab.research.google.com/github/openvinotoolkit/openvino_notebooks/blob/main/notebooks/401-object-detection-webcam/401-object-detection.ipynb#Table-of-contents:

[3] Openvino 2024.4 — openvino™ documentation. [Online]. Available: https://docs.openvino.ai/2024/index.html

[4] Linux Foundation and Yocto Project, "Welcome to the Yocto Project Documentation." [Online]. Available: https://docs.yoctoproject.org/index.html