Complete Flow for Building an Under Vehicle Scanning System (UVSS)

Below is a comprehensive, step-by-step flow for creating your Under Vehicle Scanning System (UVSS) using the Basler a2A1920-51gcBas camera on Ubuntu with C++. This guide covers hardware setup, software implementation, image processing, and system optimization, ensuring a functional UVSS tailored to your needs.

1. Hardware Setup

- Camera Mounting:

  Mount the Basler camera to capture the vehicle's underside, typically embedded in the ground or on a platform. Ensure:

  - Protection from environmental factors (e.g., weather, dust) with a rugged enclosure.
  - Alignment to capture the full width of the vehicle undercarriage.

- Lighting Installation:

  Install bright LED lights to illuminate the undercarriage. Considerations:

  - Position lights to minimize shadows and reflections.
  - Use visible light (not infrared) due to the camera's IR cut filter.

- Sensor Installation:

  Set up sensors to detect vehicle presence and trigger the camera. Options:

  - Inductive loops: Embedded in the ground to detect metal.
  - Photoelectric sensors: Detect when a vehicle breaks a light beam.

  These sensors signal the camera to start capturing images.

2. Software Installation

- Basler pylon Software Suite:

  Install the pylon Software Suite on Ubuntu for camera control with a C++ API.

  - Download from: https://www.baslerweb.com/en/downloads/software-downloads/
  - Install command:

    sudo tar -C /opt -xzf pylon-X.X.X-linux-x86_64.tar.gz
  - Follow the pylon documentation for setup.

- OpenCV for C++:

  Install OpenCV for image processing and stitching.

  - Install commands:

    sudo apt-get update

    sudo apt-get install libopencv-dev
  - Verify installation:

    pkg-config --modversion opencv4

3. Camera Configuration

- Lens Selection:

  Choose a wide-angle lens (e.g., 4-6 mm focal length) to capture the full vehicle width (~1.8 m for cars).

- Camera Settings:

  Use the pylon Software Suite to configure:

- Resolution: 1920 x 1200 (adjust as needed).
- Frame Rate: 51 fps (or lower if sufficient).
- Exposure Time: 100-500 µs to minimize motion blur.
- Trigger Mode: Enable external triggering linked to sensors.
Test settings with the pylon Viewer tool.

4. Vehicle Detection

- Sensor Integration:

Connect sensors (e.g., via GPIO on a Raspberry Pi or Ubuntu machine) to:
- Detect vehicle presence.
- Send a signal to trigger the camera.

- Trigger Mechanism:

Configure the camera to capture images when triggered. Example pylon C++ code:

```
#include <pylon/PylonIncludes.h>
using namespace Pylon;
// Initialize camera and set trigger mode
CInstantCamera camera;
camera.TriggerMode.SetValue("On");
```

5. Image Capture

- Frame Capture Strategy:

Capture multiple frames as the vehicle passes, ensuring:
- 20-30% overlap between frames for stitching.
- At 5 km/h (~1.4 m/s) and 51 fps, frames are ~2.7 cm apart—adjust speed or frame rate for better overlap.

- Capture Code:

Use the pylon C++ API to grab images. Basic example:

```
#include <pylon/PylonIncludes.h>
using namespace Pylon;
int main() {
    PylonInitialize();
    CInstantCamera camera(CTlFactory::GetInstance().CreateFirstDevice());
    camera.StartGrabbing();
    CImageFormatConverter converter;
    CPylonImage image;
    while (camera.IsGrabbing()) {
        camera.RetrieveResult(5000, ptrGrabResult);
        converter.Convert(image, ptrGrabResult);
        // Save or process image
    }
    return 0;
}
```

Refer to: https://docs.baslerweb.com/pylon-cpp

6. Image Processing

- Undistortion:

Correct lens distortion using OpenCV:

```
#include <opencv2/opencv.hpp>
cv::Mat undistortImage(cv::Mat rawImage, cv::Mat cameraMatrix, cv::Mat distCoeffs) {
    cv::Mat undistorted;
    cv::undistort(rawImage, undistorted, cameraMatrix, distCoeffs);
    return undistorted;
}
```

- Stitching:

Stitch frames into a complete undercarriage image using OpenCV:

- Detect features (e.g., SIFT, ORB).
- Compute homography and align frames.
- Example:

```
#include <opencv2/stitching.hpp>
std::vector<cv::Mat> frames;
cv::Ptr<cv::Stitcher> stitcher = cv::Stitcher::create();
cv::Mat stitched;
stitcher->stitch(frames, stitched);
```

Ensure sufficient overlap; adjust vehicle speed or frame rate if stitching fails.

7. Anomaly Detection

- Basic Detection:

Compare the stitched image to a reference using OpenCV:

```
cv::Mat diff;
cv::absdiff(stitchedImage, referenceImage, diff);
cv::threshold(diff, diff, 50, 255, cv::THRESH_BINARY);
```

- Advanced Detection:

Use machine learning (e.g., CNN) with TensorFlow or PyTorch for threat detection, requiring training data.

- Alert Mechanism:

Trigger alerts for anomalies (e.g., log to file, sound alarm).

8. Integration with Other Systems

- Automatic Number Plate Recognition (ANPR):

Add ANPR to identify vehicles using OpenCV or dedicated libraries, linking to undercarriage images.

- Data Storage:

Save stitched images and results:

```
cv::imwrite("undercarriage_" + timestamp + ".png", stitchedImage);
```

- User Interface:

Create a UI with Qt or OpenCV for real-time display.

9. Optimization

- Real-Time Performance:
  Optimize for speed:
  - Reduce resolution if acceptable.
  - Use multi-threading or OpenCV's CUDA module for GPU acceleration.
- Image Quality:
  Fine-tune lighting and exposure for varying conditions.

10. Testing and Calibration

- Test Scenarios:
  Test with different vehicles and speeds to verify coverage and stitching.
- Calibration:
  Adjust sensor timing, camera settings, and lighting based on results.
- Anomaly Testing:
  Place test objects under vehicles to ensure detection.

Additional Tips

- Documentation:
  Refer to:
  - Basler Camera: https://docs.baslerweb.com/
  - OpenCV: https://docs.opencv.org/
- Inspiration:
  Study systems like Vehant NuvoScan or patents (e.g., US20070030349A1).
- Vehicle Speed:
  Slow speeds (5-10 km/h) improve quality—consider speed bumps.

Conclusion

This flow provides a comprehensive guide to building a UVSS with a Basler camera on Ubuntu using C++. Iterate on your design for real-time performance and stitching accuracy.