Início a Programação

O Limelight suporta os protocolos REST/HTTP, Websocket, Modbus e NetworkTables para dados de mira, dados de status e configuração ao vivo. Formatos de saída JSON, Protobuf e bruto estão disponíveis. Consulte a seção de APIs da documentação para obter mais informações.

Para equipes de FRC, o protocolo recomendado é o NetworkTables. O Limelight envia todos os dados de mira, incluindo um despejo completo em JSON, para o NetworkTables a 100hz. As equipes também podem definir controles, como ledMode, janela de corte e mais via NetworkTables. As equipes de FRC podem usar as bibliotecas Limelight Lib Java e C++ para começar com o Limelight em segundos. Limelight Lib é a maneira mais fácil de começar.

Biblioteca Limelight:

Java, C++

Java

```
double tx = LimelightHelpers.getTX("");
```

C++

```
#include "LimelightHelpers.h"
```

```
double tx = LimelightHelpers::getTX("");
double ty = LimelightHelpers::getTY("");
```

Python

```
wip
```

NetworkTables

Java

```
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.networktables.NetworkTable;
import edu.wpi.first.networktables.NetworkTableEntry;
import edu.wpi.first.networktables.NetworkTableInstance;
```

```
NetworkTable table = NetworkTableInstance.getDefault().getTable("limelight");
NetworkTableEntry tx = table.getEntry("tx");
NetworkTableEntry ty = table.getEntry("ty");
NetworkTableEntry ta = table.getEntry("ta");

//Lê os valores periodicamente
double x = tx.getDouble(0.0);
double y = ty.getDouble(0.0);
double area = ta.getDouble(0.0);

//Manda o valor para a smart dashboard periodicamente
SmartDashboard.putNumber("LimelightX", x);
SmartDashboard.putNumber("LimelightY", y);
SmartDashboard.putNumber("LimelightArea", area);
```

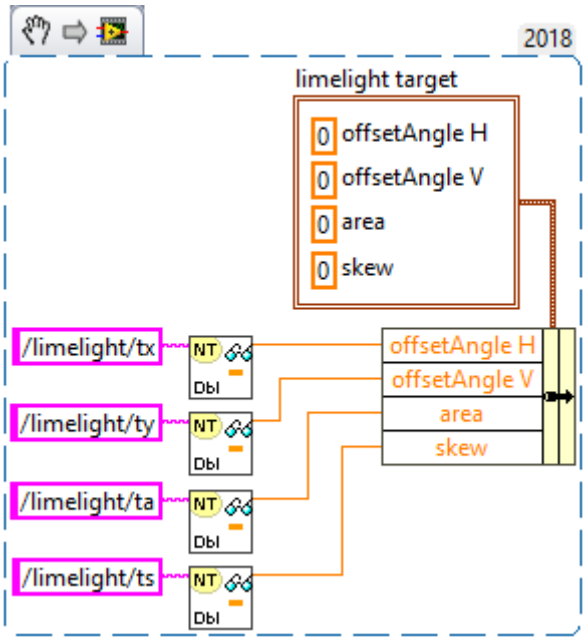
C++

```
#include "frc/smartdashboard/Smartdashboard.h"
#include "networktables/NetworkTable.h"
#include "networktables/NetworkTableInstance.h"
#include "networktables/NetworkTableEntry.h"
#include "networktables/NetworkTableValue.h"
#include "wpi/span.h"
```

```
std::shared_ptr<nt::NetworkTable> table =
nt::NetworkTableInstance::GetDefault().GetTable("limelight");
double targetOffsetAngle_Horizontal = table->GetNumber("tx",0.0);
```

```
double targetOffsetAngle_Vertical = table->GetNumber("ty",0.0);
double targetArea = table->GetNumber("ta",0.0);
double targetSkew = table->GetNumber("ts",0.0);
```

LabVIEW



Python

```
import cv2
import numpy as np

# runPipeline() é chamado todo frame pelo backend da Limelight
def runPipeline(image, llrobot):
    # converte a imagem de input para o espaço de cor do HSV
    img_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # converte o hsv para uma imagem binaria removendo qualquer pixel
    # que não sejam de acordo com os seguintes valores minimos e máximos do HSV
    img_threshold = cv2.inRange(img_hsv, (60, 70, 70), (85, 255, 255))

    # ache contornos na nova imagem binaria
    contours, _ = cv2.findContours(img_threshold,
    cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    largestContour = np.array([[]])
```

```
    # inicializa um array vazio para mandar devolta ao robô
    llpython = [0,0,0,0,0,0,0,0]

    # se os contornos foram detectados, desenhe eles
    if len(contours) > 0:
        cv2.drawContours(image, contours, -1, 255, 2)
        # grava o maior contorno
        largestContour = max(contours, key=cv2.contourArea)

        # pega a caixa delimitadora não rotacionada que envolve o contorno
        x,y,w,h = cv2.boundingRect(largestContour)

        # desenha a caixa delimitadora não rotacionada
        cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,255),2)

        # grave alguns dados customizados para mandar devolta ao robô
        llpython = [1,x,y,w,h,9,8,7]

    #retorna o maior contor para a mira da LL, a imagem modificada, e os dados customizados
    return largestContour, image, llpython
```

Revisão #5
Criado 2023-12-15 10:45:31 UTC por Enzo
Atualizado: 2023-12-18 12:08:21 UTC por Luca Carvalho